Theoretical Computer Science 134 (1994) 329–363
Elsevier

Finite-memory automata*

Michael Kaminski

Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

Nissim Francez

Department of Computer Science, Technion – Israel Institute of Technology, Technion-city, Haifa, 32000, Israel

Communicated by A.R. Meyer
Received October 1993

Abstract

Kaminski, M., and N. Francez, Finite-memory automata, Theoretical Computer Science 134 (1994) 329–363.

A model of computation dealing with *infinite alphabets* is proposed. This model is based on replacing the equality test by *substitution*. It appears to be a natural generalization of the classical Rabin–Scott finite-state automata and possesses many of their closure and decision properties. Also, when restricted to finite alphabets the model is equivalent to finite-state automata.

1. Introduction

In this paper we introduce a model of computation dealing with *infinite alphabets*, a generalization of the classical Rabin–Scott finite-state automata [6]. In doing so, we are aiming towards a very restrictive model, capable of recognizing only the natural analog of *regular languages* over finite alphabets. In addition, we would like our model, and the class of languages recognizable by it, to enjoy as many of the properties of the family of finite automata and regular languages as possible. Thus we are interested in preserving in *closure* under Kleene operations and Boolean operations as well as in having decidable the emptyness problem. We succeed in doing so, except for closure under *complementation*, which is achievable only by passing to a restricted model.

Correspondence to: N. Francez, Department of Computer Science, Technion – Israel Institute of Technology, Technion-city, Haifa 32000, Israel. Email: francez@cs.technion.ac.il.

* The part of the second author was partially funded by a grant from the Israeli Academy of Sciences (basic research) and by the Fund for the Promotion of Research in the Technion.

SSDI 0304-3975(93)E0205-I

Clearly, our aim cannot be achieved by preserving finite-stateness in its strict meaning. On the other hand, allowing for arbitrary infinite-state automata is obviously too powerful, as it renders *every* language recognizable, by having a state corresponding to each letter in the infinite alphabet, see [2]. Thus we need a very restricted memory structure of the automaton, so that it will not be able "to take advantage" of its memory capabilities beyond what is needed for our purposes. We also want to preserve the *computability* of the recognizable languages, so the model should not be powerful enough to do *coding*, or even *counting*.

For example, the deterministic version of our automata should be similar, as an *acceptor*, to the programs defined below.[1] A program input is a sequence of *atomic* symbols over an infinite alphabet Σ, and a program itself consists of a specification of a finite set of variables v_i, $i=1,2,\ldots,r$, and a finite sequence of commands of the following type.

- $v_i := \sigma$, where $\sigma \in \Sigma$. This operation is the initialization of the variable v_i.
- **read**(v_i). This operation (substitution) assigns the value of the next input symbol to the variable v_i.
- **type**(v_i). This operation prints the value of the variable v_i.
- $v_i := v_j$. This operation assigns the value of v_j to v_i.
- **if** $v_i = v_j$, **then go to** k. This operation (equality test) passes the control to the kth operation in the sequence, if the value of v_i is equal to the value of v_j, otherwise it skips to the next command in the sequence.
- **halt**. This command terminates the program.

Thus by restricting the manipulative power of the automaton to copying and comparing only, without the ability to apply *any* modification functions, all the automaton is capable of doing is to "remember" some bounded number of previously read symbols. Therefore, recognizable languages will have the usual characteristics of regular languages.[2] Typically, it is able to detect the presence of a specific letter, or the appearance of simple patterns such as a repetition of the same letter, etc..

Roughly speaking, the basic idea behind our definition is to equip the automaton with a finite set of *proper* states (as in the classic model), in which the "real computation" is done. In addition, the automaton is equipped with a finite set of registers, each capable of being *empty*,[3] or storing a symbol from the infinite alphabet. We refer to these registers as "*windows*." The restricted power of the automaton is obtained by highly restricting its transition relation.[4] Here, the novel idea is to replace the *equality test* (of the next input symbol to some element of the finite alphabet), which underlies the classic model, by some (extended form of) *substitution*. The latter employs both equality tests and copying. What the automaton does with the next input symbol is

[1] These programs are a version of *data-independent* programs introduced in [10].

[2] Obviously, for a finite alphabet Σ the above programs are equivalent to finite transducers, and, the programs without the command **type** are just acceptors equivalent to finite automata.

[3] For technical convenience, emptiness is represented by holding a special symbol # that is not contained in the infinite alphabet.

[4] Our model is nondeterministic, so the transition is not a function.

the following. If no window contains the input symbol, then it is *copied* into a specified window (depending on the state). Otherwise, the automaton "remembers" that the input symbol has been previously read, in which case the usual equality test applies. As it turns out, relating the next input symbol to *many* windows simultaneously leads to an easier development of the theory without strengthening the model, see Section 3.

The idea of replacing equality test with substitution originates from [7], where it was applied to an infinite alphabet of a very specific structure, where the letters had the form $r(x_i, x_j)$ and were interpreted as binary relation symbols. Words over this alphabet have a structure bearing a strong relationship to *Data-log*, a useful data-base query language ([19]). Other useful interpretations of infinite alphabets are not hard to imagine. Finite sequences of simple patterns occur in many contexts within computer science. *Actions* of concurrent processes, when concurrency and communication are restricted to very simple patterns, are another possible interpretation of infinite alphabets.

A beneficial by-product of our theory is an ability to represent in a more compact way some classical finite-state automata, in case the finite alphabet is so large to best treated as if it were infinite. Thus *process identifiers* for example (in computer networks) will usually be natural numbers. Any practical network will have some bound on these numbers. However, such bounds may be determined in some complicated, architecture-dependent way, and at a higher level of abstraction it is best to ignore these bounds and consider arbitrary natural numbers.

An important facet of our theory is a certain *indistinguishability* view of the finite alphabet embedded in the modus operandi of the automaton. Languages are only unique up to *automorphisms* of the alphabet.[5] Thus the *actual* letters occurring in the input are of no real significance. Only the initial and repetition patterns matter. This follows from the nature of substitution. If a *new* letter (i.e., one not in any window) is copied and later successfully compared, any other new letter, having appeared in the same position, would cause the same transitions. This, in particular, implies that the usual pumping lemma does not hold in our model.

One can try to extend our theory to other models of computation, such as automata on infinite words and pushdown automata. Here, we restricted the presentation to the (analog of the) regular case only. However, the results have nontrivial proofs and establish the basic techniques that might be applicable to such extensions.

The paper is organized as follows. In the next section we define the model of computation and present some of its basic properties. Section 3 deals with various closure properties, and in Section 4 we consider the deterministic and two-way models. The last section contains a concluding discussion and a list of open problems which seem to us to be of interest. We conclude the paper with an appendix which contains a decidability result. All proofs in this paper are constructive.

[5] Actually, only automorphisms which keep fixed those symbols to which windows are initialized are considered.

2. Definitions and basic properties

In this section we define the model of computation and show some of its basic properties.

Let Σ be an infinite alphabet and let $\#$ be a symbol not belonging to Σ. An *assignment* is a word $w_1 w_2 \dots w_r \in (\Sigma \cup \{\#\})^*$ such that if $w_i = w_j$ and $i \neq j$, then $w_i = \#$. That is, an assignment is a word over $\Sigma \cup \{\#\}$ where each symbol from Σ appears at most one time. According to the informal description of the model presented in the introduction, assignments represent the contents of the registers of the automaton: the symbol in the ith register (window) is w_i. If $w_i = \#$, then the ith window is empty. In our model we assume that a symbol cannot simultaneously appear in two or more windows. This restriction does not weaken the model, see Definition 2 and Theorem 2 in the next section.

For a word $\boldsymbol{w} = w_1 w_2 \dots w_r \in (\Sigma \cup \{\#\})^*$ we define the *contents* of $\boldsymbol{w}$, denoted $[\boldsymbol{w}]$, by $[\boldsymbol{w}] = \{w_i : i = 1, 2, \dots, r\}$, i.e., $[\boldsymbol{w}]$ consists exactly of those symbols of $\Sigma \cup \{\#\}$ which appear in $\boldsymbol{w}$.

Definition 1. A *finite-memory automaton* is a system $\boldsymbol{A} = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F \rangle$, where

- S is a finite set of *states*.
- $q_0 \in S$ is the *initial* state.
- $\boldsymbol{u} = w_1 w_2 \dots w_r \in (\Sigma \cup \{\#\})^r$ is the *initial* assignment – registers' initialization.
- $\rho : S \to \{1, 2, \dots, r\}$ is a partial function from S to $\{1, 2, \dots, r\}$ called the *reassignment*. The intuitive meaning of ρ is as follows. If $\boldsymbol{A}$ is in state s, $\rho(s)$ is defined, and the input symbol appears in no window, then $\boldsymbol{A}$ "forgets" the contents of the $\rho(s)$th window and copies the input symbol into that window.
- $\mu \subseteq S \times \{1, 2, \dots, r\} \times S$ is the *transition* relation. The intuitive meaning of μ is as follows. If $\boldsymbol{A}$ is in state s, the input symbol is equal to the contents of the kth window, and $(s, k, t) \in \mu$, then $\boldsymbol{A}$ may enter state t. In addition, if the input symbol appears in no window and is placed into the kth window ($k = \rho(s)$), then in order to enter state t the transition relation must contain (s, k, t). That is, the reassignment is made prior to a transition.
- $F \subseteq S$ is the set of *final* states.

The initial assignment of automaton $\boldsymbol{A}$ and its length are denoted by $\boldsymbol{u}_{\boldsymbol{A}}$ and $r_{\boldsymbol{A}}$, respectively. That is, $\boldsymbol{u} = \boldsymbol{u}_{\boldsymbol{A}}$ and $r = r_{\boldsymbol{A}}$.

Similar to the case of finite automata, $\boldsymbol{A}$ can be represented by its initial assignment and a directed graph whose nodes are states. There is an edge from s to t, if there exists k such that $(s, k, t) \in \mu$. Such edge is labeled k. Also, if for a node s the value of ρ is defined, then s is labeled $\rho(s)$. For graph representation of finite-memory automata, see Example 1 below.

An actual state of A is a state of S together with the contents of all windows. Thus A has infinitely many states[6] which are pairs (s, w), where $s \in S$ and w is an assignment of length r. These are called the *configurations* of A. The set of all configurations of A is denoted by S^c. The pair $q_0^c = (q_0, u)$ is called the *initial* configuration, and the configurations with the first component in F are called *final* configurations. The set of final configurations is denoted by F^c.

The transition relation μ induces the following relation μ^c on $S^c \times \Sigma \times S^c$.

Let $w = w_1 w_2 \dots w_r$ and $v = v_1 v_2 \dots v_r$. Then $((s, w), \sigma, (t, v)) \in \mu^c$ if and only if the two following conditions are satisfied.

- If $\sigma = w_k \in [w]$, then $v = w$ and $(s, k, t) \in \mu$.
- If $\sigma \notin [w]$, then $\rho(s)$ is defined, $v_{\rho(s)} = \sigma$, for each $k \neq \rho(s)$, $v_k = w_k$, and $(s, \rho(s), t) \in \mu$.

Let $\sigma = \sigma_1 \sigma_2 \dots \sigma_n$ be a word over Σ. A *run* of A on σ consists of a sequence of configurations $c_0, c_1, \dots, c_n$ such that $c_0 = q_0^c$ and $(c_{i-1}, \sigma_i, c_i) \in \mu^c$, $i = 1, 2, \dots, n$.

We say that A *accepts* $\sigma \in \Sigma^*$, if there exists a run $c_0, c_1, \dots, c_n$ of A on σ such that $c_n \in F^c$. The set of all words accepted by A is denoted by $L(A)$ and is referred to as a *quasi-regular* set.

Before considering some general properties of finite-memory automata, we present several examples which show the difference and similarity between the models over finite and infinite alphabets.

Example 1. Consider a finite-memory automaton $A = \langle \{q_0, q, f\}, q_0, \#\#, \rho, \mu, \{f\} \rangle$, where

- $\rho(q_0) = 1$, $\rho(q) = \rho(f) = 2$; and
- $\mu = \{(q_0, 1, q_0), (q_0, 1, q), (q, 1, f), (q, 2, q), (f, 1.f), (f, 2, f)\}$.

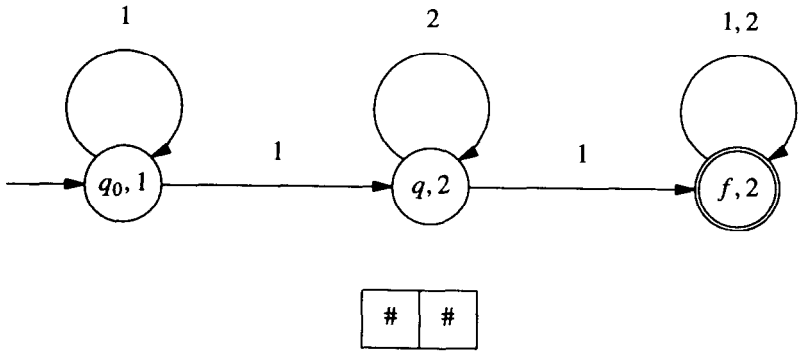
This automaton has the graph representation shown in Fig. 1.

One can easily verify that the language $L_1 = L(A)$ consists exactly of those words where some element of Σ appears twice or more:

$$L_1 = \{\sigma_1 \sigma_2 \dots \sigma_n : \text{there exist } 1 \leqslant i \leqslant j \leqslant n \text{ such that } \sigma_i = \sigma_j\}.$$

The behavior of A on such words is as follows. Being the initial state, the automaton stores new input symbols in the first window. When reading a symbol that appears twice, A changes the state to q and, storing new input symbols in the second window, waits for the second appearance of the symbol stored in the first window. Then it enters the final state. For example, $abcbd \in L_1$, because b appears twice in that word. An accepting run of A on $abcbd$ is $(q_0, \#\#), (q_0, a\#), (q, b\#), (q, bc), (f, bc), (f, bd)$. We shall see in the sequel that the complement of L_1 (with respect to Σ^*) is not quasi-regular.

[6] This is the major difference between finite and finite-memory automata. In particular, since finite-memory automata have infinitely many actual states, the pumping lemma does not hold for quasi-regular languages, see Example 3. For the same reason the computation power of the deterministic and two-way models differs from that of the one-way nondeterministic one, see Section 4.



initialization

Fig. 1.

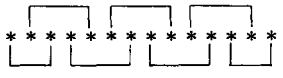
Example 2. *Let* $\Sigma' = \{\sigma_1, \sigma_2, \ldots, \sigma_r\}$ be an r-element subset of Σ and let $A' = \langle S, q_0, \mu', F\rangle$ be a finite automaton[7] over Σ'. Consider a finite-memory automaton $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, \boldsymbol{F}\rangle$, where $\boldsymbol{u} = \sigma_1\sigma_2 \ldots \sigma_r$, the reassignment ρ is nowhere defined and $(s, k, t) \in \mu$ if and only if $(s, \sigma_k, t) \in \mu'$. It immediately follows that $L(A) = L(A')$. Therefore every regular language is quasi-regular.

The next example shows that the pumping lemma does not hold for quasi-regular languages.

Example 3. Let A be a finite-memory automaton defined by the diagram of Fig. 2.

Notice that the reassignment function is not defined for either of q_2 and q_4. Therefore the automaton can leave q_2 if and only if the input symbol appears in the first window, and can leave q_4 if and only if the input symbol appears in the second window.

Let $n \geqslant 1$ and let $\tau_0, \tau_1, \ldots, \tau_{2n}$ be distinct elements of Σ. Consider a word $\sigma = \sigma_1\sigma_2 \ldots \sigma_{4n+2}$, where $\sigma_1 = \sigma_3 = \tau_0$, $\sigma_{4n} = \sigma_{4n+2} = \tau_{2n}$, and $\sigma_{2i} = \sigma_{2i+3} = \tau_i$ for $i = 1, 2, \ldots, 2n-1$. That, is σ is of the form



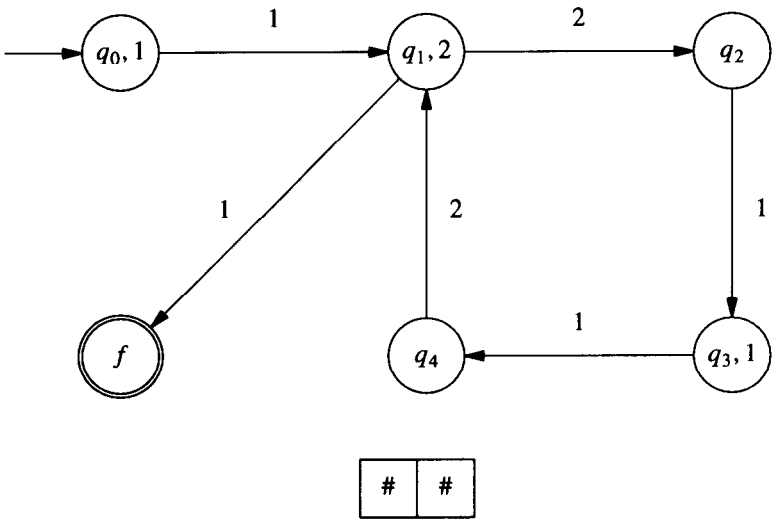
where brackets connect the same τ_i's. A straightforward induction shows that $c_0, c_1, \ldots, c_{4n+2}$, where

$$c_0 = (q_0, \#\#),\ c_1 = (q_1, \tau_0\#),$$

$$c_{4i+2} = (q_2, \tau_{2i}\tau_{2i+1}),\ c_{4i+3} = (q_3, \tau_{2i}\tau_{2i+1}),\ c_{4i+4} = (q_4, \tau_{2i+2}\tau_{2i+1}),$$

$$c_{4i+5} = (q_1, \tau_{2i+2}\tau_{2i+1}),\ i = 0, 1, \ldots, n-1, \text{ and } c_{4n+2} = (f, \tau_{2n-1}\tau_{2n}),$$

[7] That is, S is a finite set of states, $q_0 \in S$ is the initial state, $\mu' \subseteq S \times \Sigma' \times S$ is the transition relation, and $F \subseteq S$ is the set of final states. The sequence of states $s_0, s_1, \ldots, s_n$ is an accepting run on A on the word $\sigma_1\sigma_2 \ldots \sigma_n$, if $s_0 = q_0$, $s_n \in F$, and $(s_{i-1}, \sigma_i, s_i) \in \mu'$, $i = 1, 2, \ldots, n$.



initialization

Fig. 2.

is an accepting run on A on σ. Thus $\sigma \in L(A)$. We claim that σ contains no nonempty pattern that may be pumped. To prove our claim, assume to the contrary, that $\sigma = xyz$, where $y = \sigma_l \sigma_{l+1} \dots \sigma_{m-1}$ is nonempty and for some $j > 1$, $xy^j z \in L(A)$. An inspection of the automaton's diagram shows that every word accepted by A must be of the length $4k+2$, $k = 0, 1, \dots$ Therefore $m-l$ is a positive multiple of 4, which implies $m-l \geqslant 4$. Then $xy^j z$ contains the pattern $\sigma_{m-2}\sigma_{m-1}\sigma_l\sigma_{l+1}\sigma_{l+2}$. Let $(s_0, w_0), (s_1, w_1), \dots, (s_{m+2}, w_{m+2})$ be a run of A on $\sigma_1\sigma_2 \dots \sigma_{m-2}\sigma_{m-1}\sigma_l\sigma_{l+1}\sigma_{l+2}$. Since on each input symbol, from each configuration, A can make at most one move, and σ and $xy^j z$ have the same prefixes of length $m-1$, $(s_{m-1}, w_{m-1}) = c_{m-1}$. We distinguish between the cases of $m-1 = 4i+2$, $m-1 = 4i+3$, $m-1 = 4i+4$, and $m-1 = 4i+5$.

Assume $m-1 = 4i+2$. Then $(s_{m-1}, w_{m-1}) = c_{m-1} = c_{4i+2} = (q_2, \tau_{2i}\tau_{2i+1})$. The automaton can leave q_2 if and only if $\sigma_l = \tau_{2i}$, which, by the definition of σ, holds if and only if either $l = 4i = m-3$ or $l = 4i+3 = m$. Either of the equalities leads to a contradiction, because $m-l \geqslant 4$.

Assume $m-1 = 4i+3$. Then $(s_{m-1}, w_{m-1}) = c_{m-1} = c_{4i+3} = (q_3, \tau_{2i}\tau_{2i+1})$, implying $(s_m, w_m) = (q_4, \sigma_l\tau_{2i+1})$. The automaton can leave q_4 if and only if $\sigma_{l+1} = \tau_{2i+1}$. By the definition of σ, the equality holds if and only if either $l+1 = 4i+2 = m-2$ or $l+1 = 4i+5 = m+1$. Since $m-l \geqslant 4$, this is impossible.

Assume $m-1 = 4i+4$. Then $(s_{m-1}, w_{m-1}) = c_{m-1} = c_{4i+4} = (q_4, \tau_{2i+2}\tau_{2i+1})$. The automaton can leave q_4 if and only if $\sigma_l = \tau_{2i+1}$, which is impossible.

Assume $m-1 = 4i+5$. Then $(s_{m-1}, w_{m-1}) = c_{m-1} = c_{4i+5} = (q_1, \tau_{2i+2}\tau_{2i+1})$, implying $(s_m, w_m) = (q_2, \tau_{2i+2}\sigma_l)$. The automaton can leave q_2 if and only if $\sigma_l = \tau_{2i+2}$, which is impossible.

Thus $xy^jz \notin L(A)$, which shows that $L(A)$ does not satisfy the pumping lemma for regular languages.

Since the restriction of the set of configurations to a finite alphabet is finite, we have the following result.

Proposition 1. *Let $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F \rangle$ be a finite-memory automaton and let Σ' be a finite subset of Σ. Then $L(A) \cap \Sigma'^*$ is a regular language (over Σ').*

Proof. Consider a finite automaton $A' = \langle S', q_0', \mu', F' \rangle$ over Σ' that is defined as follows.

- $S' = S^c \cap (S \times (\Sigma' \cup [\boldsymbol{u}] \cup \{\#\})^{r+1})$. Since Σ' is finite, S' is finite as well.
- $q_0' = (q_0, \boldsymbol{u})$.
- $\mu' = \mu^c \cap (S' \times \Sigma' \times S')$.
- $F' = F^c \cap S'$.

Let $\sigma \in \Sigma'^*$. It immediately follows from the construction above that each accepting run of A on σ is an accepting run of A' on σ, and vice versa. Thus $\sigma \in L(A) \cap \Sigma'^*$ if and only if $\sigma \in L(A')$. □

Propositions 2 and 3 reflect the fact that a finite-memory automaton can only sense "new" input symbols, i.e., ones appearing in the contents of the most recent assignment. It cannot, however, distinguish between different "new" symbols. The above property of finite-memory automata is useful a tool for proving that a language is not quasi-regular, see Example 4 below.[8] Note that an occurrence of an input symbol that belonged to a previous assignment and was "forgotten" in a reassignment is also "new." We need the following auxiliary result.

Lemma 1. *Let $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F \rangle$ be a finite-memory automaton. Then for each automorphism $\iota: \Sigma \to \Sigma$ we have $\iota(L(A)) = L(A_{(q_0, \iota(\boldsymbol{u}))})$, where $A_{(q_0, \iota(\boldsymbol{u}))} = \langle S, q_0, \iota(\boldsymbol{u}), \rho, \mu, F \rangle$*[9].

Proof. We contend that $(s_0, \boldsymbol{w}_0), (s_1, \boldsymbol{w}_1), \ldots, (s_n, \boldsymbol{w}_n)$ is a run of A on σ if and only if $(s_0, \iota(\boldsymbol{w}_0)), (s_1, \iota(\boldsymbol{w}_1)), \ldots, (s_n, \iota(\boldsymbol{w}_n))$ is a run of $A_{(q_0, \iota(\boldsymbol{u}))}$ on $\iota(\sigma)$. First we prove by induction on the length of σ, denoted by n, that if $(s_0, \boldsymbol{w}_0), (s_1, \boldsymbol{w}_1), \ldots, (s_n, \boldsymbol{w}_n)$ is a run of A on σ, then $(s_0, \iota(\boldsymbol{w}_0)), (s_1, \iota(\boldsymbol{w}_1)), \ldots, (s_n, \iota(\boldsymbol{w}_n))$ is a run of $A_{(q_0, \iota(\boldsymbol{u}))}$ on $\iota(\sigma)$. Obviously, this is true if σ is the empty word. Assume that the claim holds for all words of length n. Let $(s_0, \boldsymbol{w}_0), (s_1, \boldsymbol{w}_1), \ldots, (s_n, \boldsymbol{w}_n), (s_{n+1}, \boldsymbol{w}_{n+1}))$ be a run of A on $\sigma\sigma$. We have to prove that $(s_0, \iota(\boldsymbol{w}_0)), (s_1, \iota(\boldsymbol{w}_1)), \ldots, (s_n, \iota(\boldsymbol{w}_n)), (s_{n+1}, \iota(\boldsymbol{w}_n))$ is a run of $A_{(q_0, \iota(\boldsymbol{u}))}$ on $\iota(\sigma)\iota(\sigma)$. Since, by the induction hypothesis, $(s_0, \iota(\boldsymbol{w}_0)), (s_1, \iota(\boldsymbol{w}_1)), \ldots, (s_n, \iota(\boldsymbol{w}_n))$ is a run of $A_{(q_0, \iota(\boldsymbol{u}))}$ on $\iota(\sigma)$, it suffices to show that $((s_n, \iota(\boldsymbol{w}_n)), \iota(\sigma), (s_{n+1}, \iota(\boldsymbol{w}_{n+1}))) \in \mu^c$.

[8] This property is also used in the appendix for proving the decidability of an instance of the inclusion problem for quasi-regular sets.

[9] Here we implicitly extended ι to $\Sigma \cup \{\#\}$ by putting $\iota\{\#\} = \#$.

If for some $k=1,2,\ldots,r_A$, $\sigma=w_k\in[w_n]$, then $(s_n, k, s_{n+1})\in\mu$, and $w_{n+1}=w_n$. Therefore $\iota(\sigma)$ appears in the kth position of $\iota(w_n)$. Hence $((s_n, \iota(w_n)), \iota(\sigma), (s_{n+1}, \iota(w_{n+1})))\in\mu^c$.

If $\sigma\notin[w_n]$, then $(s_n, \rho(s_n), s_{n+1})\in\mu$, and w_{n+1} is obtained from w_n by replacing the content of the $\rho(s_n)$th window with σ. Since $\iota(\sigma)\notin[\iota(w_n)]$, $\iota(w_{n+1})$ is obtained from $\iota(w_n)$ by replacing the content of the $\rho(s_n)$th window with $\iota(\sigma)$. Again, $((s_n, \iota(w_n)), \iota(\sigma), (s_{n+1}, \iota(w_{n+1})))\in\mu^c$.

Now let $(s_0, \iota(w_0)), (s_1, \iota(w_1)), \ldots, (s_n, \iota(w_n))$ be a norm of $A_{(q_0, \iota(u))}$ on $\iota(\sigma)$. By the "only if" part of the proof applied to the inverse automorphism ι^{-1} we obtain that $(q_0, u), (s_1, w_1), \ldots, (s_n, w_n)$ is a run of A on σ, which completes the proof. □

Proposition 2 (Closure under automorphisms). *Let $A=\langle S, q_0, u, \rho, \mu, F\rangle$ be a finite-memory automaton. Then for each automorphism $\iota:\Sigma\to\Sigma$ that is an identity on $[u]$ and each $\sigma\in\Sigma^*$, $\sigma\in L(A)$ if and only if $\iota(\sigma)\in L(A)$.*

Proof. The result immediately follows from Lemma 1, because, in the conditions of Proposition 2, the automata $A_{(q_0, \iota(u))}$ and A coincide. □

Proposition 3 (Indistinguishability property of finite-memory automata). *Let $A=\langle S, q_0, u, \rho, \mu, F\rangle$ be a finite-memory automaton. If $xy\in L(A)$, then there exists a subset $\Sigma'\subseteq[x]$ such that the cardinality of Σ' does not exceed r_A and the following holds. For any $\sigma\notin\Sigma'$ and any $\tau\notin[y]\cup\Sigma'$, the word $x(y(\sigma\,|\,\tau))$ obtained from xy by the substitution of τ for each occurrence of σ in y belongs to $L(A)$.*

Proof. Let x be a word of length i and let $(s_0, w_0), (s_1, w_1), \ldots, (s_n, w_n)$ be an accepting run of A on xy. Let $\Sigma'=[w_i]$, $\sigma\notin[w_i]$, and $\tau\notin[y]\cup\Sigma'$.

In order to prove that $x(y(\sigma\,|\,\tau))\in L(A)$, it suffices to show that $y(\sigma\,|\,\tau)\in L(A_{(s_i, w_i)})$, where $A_{(s_i, w_i)}=\langle S, s_i, w_i, \rho, \mu, F\rangle$. Let ι be the automorphism of Σ that permutes σ with τ and leaves fixed all other symbols. Then $y(\sigma\,|\,\tau)=\iota(y)$, and the result follows from Proposition 2, because neither σ nor τ belongs to $[w_i]$. □

Example 4. Consider a language L_2 that consists of all words whose last symbol is different from all others. That is,

$$L_2=\{\sigma_1\sigma_2\ldots\sigma_n\colon \sigma_i\neq\sigma_n,\ i=1,2,\ldots,n-1\}.$$

We contend that L_2 is not quasi-regular. To prove our contention, assume to the contrary that L_2 is accepted by an r-window finite-memory automaton A. Let $x=\sigma_1\sigma_2\ldots\sigma_r\sigma_{r+1}$ and $y=\sigma_{r+2}$, where all σ_i's are distinct. Then $xy\in L_2$ $(=L(A))$. Let Σ' be a subset of $[x]$ provided by Proposition 3. Since the cardinality of Σ' does not exceed r, there exists an $i\in\{1,2,\ldots,r+1\}$ such that $\sigma_i\notin\Sigma'$. Since $[x]\cap[y]=\emptyset$, it follows that $\sigma_i\notin[y]\cup\Sigma'$. By Proposition 3, $x(y(\sigma_{r+2}\,|\,\sigma_i))\in L(A)$. However in the last word the symbol σ_i appears both in the ith and the last positions. This contradicts the assumption $L(A)=L_2$.

Being unable to distinguish between different new input symbols, sometimes a finite-memory automaton cannot distinguish between a new symbol and a symbol stored in one of its windows. Such a situation occurs when a new input symbol replaces an "old" one. Then the behavior of the automaton is exactly like when it reads the symbol stored in the "reassignment" window. We illustrate this property of finite-memory automata by the following proposition.

Proposition 4. *If A accepts a word of length n, then it accepts a word of length n that contains at most r_A distinct symbols.*

Proof. Let $\sigma=\sigma_1\sigma_2\dots\sigma_n\in L(A)$ and let $R=(s_0, w_0), (s_1, w_1), \dots, (s_n, w_n)$, where $w_i=w_{i,1}\dots w_{i,r_A}$, $i=1,2,\dots,n$, be a run of A and p. Let $m_R(\sigma)$ be the minimal integer i such that $\sigma_i\neq[w_{i-1}]$ and $w_{i-1,\rho(s_{i-1})}\neq\#$, if such an i exists, and be ∞ otherwise. That is, if $m_R(\sigma)=i<\infty$, then σ_i is "new relatively to $[w_{i-1}]$" and is placed into a nonempty window.

It follows that if σ contains more than r_A distinct symbols, then $m_R(\sigma)<\infty$. Therefore in order to prove the proposition it suffices to show that if $m_R(\sigma)<\infty$ for some accepting run R of A on p, then there exists a word σ' of length n and an accepting run R' of A and σ' such that $m_R(\sigma)<m_{R'}(\sigma')$.

So, let R be an accepting run of A on σ such that $m_R(\sigma)=i<\infty$. Let ι be an automorphism of Σ such that $\iota(\sigma_i)=w_{i-1,\rho(s_{i-1})}$, $\iota(w_{i-1,\rho(s_{i-1})})=\sigma_i$, and $\iota(\sigma)=\sigma$ for $\sigma\neq\sigma_i, w_{i-1,\rho(s_{i-1})}$. We contend that $R'=(q_0, u), (s_1, w_1), \dots, (s_{i-1}, w_{i-1}), (s_i, \iota(w_i)), \dots, (s_n, \iota(w_n))$ is an accepting run of A on $\sigma'=\sigma_1\dots\sigma_{i-1}\iota(\sigma_i\dots\sigma_n)$ such that $m_R(\sigma)<m_{R'}(\sigma')$.

Since R is a run of A on σ, in order to prove that R' is an accepting run of A on σ' it suffices to show that $(s_{i-1}, w_{i-1}), \iota(\sigma_i), (s_i, \iota(w_i))\in\mu^c$ and that $(s_i, \iota(w_n)), \dots, (s_n, \iota(w_n))$ is an accepting run of $A_{(s_i,\iota(w_i))}=\langle S, s_i, \iota(w_i), \rho, \mu, F\rangle$ on $\iota(\sigma_{i+1}\dots\sigma_n)$. Since σ_i is stored in the $\rho(s_{i-1})$th window, $(s_{i-1}, \rho(s_{i-1}), s_i)\in\mu$, and the relation $((s_{i-1}, w_{i-1}), \iota(\sigma_i), (s_i, \iota(w_i))\in\mu^c$ follows from $\iota(\sigma_i)=w_{i-1,\rho(s_{i-1})}$. Since $(s_i, w_i), \dots, (s_n, w_n)$ is an accepting run of $A_{(s_i,w_i)}=\langle S, s_i, w_i, \rho, \mu, F\rangle$ on $\sigma_{i+1}\dots\sigma_n$, the fact that $(s_i, \iota(w_i)), \dots, (s_n, \iota(w_n))$ is an accepting run of $A_{(s_i,\iota(w_i))}=\langle S, s_i, \iota(w_i), \rho, \mu, F\rangle$ on $\iota(\sigma_{i+1}\dots\sigma_n)$ immediately follows from the proof of Lemma 1. Finally, the inequality $m_{R'}(\sigma')>m_R(\sigma)$ follows from $\iota(\sigma_i)=w_{i-1,\rho(s_{i-1})}$. This completes the proof of Proposition 4. □

Remark 1. It follows from the proof of Proposition 4 that if u has l empty windows, then A accepts a word of length n that contains at most l distinct symbols not belonging to $[u]$.

Using the indistinguishability property of finite-memory automata given by Proposition 4, we can easily prove that the emptiness problem for finite-memory automata is decidable.

Theorem 1. *It is decidable whether a quasi-regular language is empty.*

Proof. Let A be a finite-memory automaton and let $\Sigma' = [u_A] \cup \{\sigma_1, \ldots, \sigma_l\}$ be a subset of Σ of cardinality r_A such that $[u_A] \cap \{\sigma_1, \ldots, \sigma_l\} = \emptyset$. We contend that $L(A) \neq \emptyset$ if and only if $L(A) \cap \Sigma'^* \neq \emptyset$. The "if" part is immediate. Let $L(A) \neq \emptyset$. By Remark 1, there exists a subset $\Sigma'' = [u] \cup \{\tau_1, \ldots, \tau_l\}$ of Σ such that $L(A) \cap \Sigma''^* \neq \emptyset$. Let ι be an automorphism of Σ such that $\iota(\sigma_i) = \tau_i$, $\iota(\tau_i) = \sigma_i$ for $i = 1, \ldots, l$, and $\iota(\sigma) = \sigma$ for $\sigma \neq \sigma_i, \tau_i$. Then,

$$L(A) \cap \Sigma'^* = L(A) \cap \iota(\Sigma''^*) = \iota(L(A) \cap \Sigma''^*),$$

where the second equality follows from Proposition 2. Since $L(A) \cap \Sigma''^* \neq \emptyset$, $L(A) \cap \Sigma'^* \neq \emptyset$ as well.

Now the decidability of the emptiness problem follows from the above contention and Proposition 1. □

Proposition 5. *Quasi-regular sets are not closed under complementation.*

Proof. Let L_1 be the quasi-regular language from Example 1. Then $\bar{L}_1$ – the complement of L_1 to Σ^* consists of all words where each symbol appears at most one time. We contend that this language is not quasi-regular. Assume to the contrary that there exists a finite-memory automaton A' such that $\bar{L}_1 = L(A')$. Since Σ is infinite, there exists a word $\sigma \in L(A')$ of length $r_{A'} + 1$. By Proposition 4, A' accepts a word σ' of length $r_{A'} + 1$ that contains at most $r_{A'}$ distinct symbols. Therefore some symbol of Σ must appear in σ' more than one time, in contradiction with the assumption $\bar{L}_1 = L(A')$. □

Remark 2. The proof of Proposition 5 can be easily extended to show that no machine which is able to remember only a fixed number of symbols accepts $\bar{L}_1$. Therefore nonclosure under complementation might, in some sense, seem to be a "natural" property of families of languages containing L_1 and defined by simple machines over infinite alphabets. It should be mentioned, however, that finite-memory automata have a natural deterministic analog, and the language accepted by deterministic automata are closed under complementation (since we can simply replace F by $S - F$). Thus the classes of languages accepted by deterministic and nondeterministic finite-memory automata are different. Deterministic finite-memory automata are considered in more detail in Section 4.

3. Closure properties of quasi-regular languages

In this section we consider closure properties of quasi-regular languages. The proofs are based on the standard construction adapted to a slightly modified version of finite-memory automata that allows the possibility of relating the next input symbol to *many* windows simultaneously.

Definition 2. A finite-memory automaton with *multiple assignment*, or shortly M-automaton, over Σ is a system $A=\langle S, q_0, \boldsymbol{u}, \rho, \mu, F\rangle$, where

- S is a finite set of *states*.
- $q_0 \in S$ is the *initial* state.
- $\boldsymbol{u}=w_1 w_2 \ldots w_r \in (\Sigma \cup \{\#\})^r$ is the *initial* M-assignment. (Notice that an assignment for an M-automaton can be *any* word over $(\Sigma \cup \{\#\})^r$, i.e., the pairwise distinctness condition is relaxed.)
- $\rho: S \to 2^{\{1,2,\ldots,r\}} - \{\emptyset\}$ is a function from S to the set of all nonempty subsets of $\{1, 2, \ldots, r\}$, called the *M-reassignment. The intuitive meaning of ρ is as follows. If A is in state s, then it replaces the contents of the windows indexed by the elements of $\rho(s)$ by the input symbol.*
- $\mu \subseteq S \times (2^{\{1,2,\ldots,r\}} - \{\emptyset\}) \times S$ is the *M-transition* relation. The intuitive meaning of μ is as follows. If A is in state s, P is the set of indices of the windows containing the input symbol (after the reassignment is made), and $(s, P, t) \in \mu$, then A may enter state t.
- $F \subseteq S$ is the set of *final states*.

An *M-configuration* of an M-automaton A is a pair $(s, \boldsymbol{w})$, where $s \in S$ and $\boldsymbol{w} \in (\Sigma \cup \{\#\})^r$. As in the case of finite-memory automata, the set of all M-configurations of A is denoted by S^c, i.e., $S^c = S \times (\Sigma \cup \{\#\})^r$. The pair $q_0^c = (q_0, \boldsymbol{u})$ is called the *initial* M-configuration, and the elements $F \times (\Sigma \cup \{\#\})^r$ are called *final* M-configurations. The set of final M-configurations is denoted by F^c.

The relation μ induces the following relation μ^c on $S^c \times \Sigma \times S^c$.

Let $\boldsymbol{w}=w_1 w_2 \ldots w_r$ and $\boldsymbol{v}=v_1 v_2 \ldots v_r$. Then $((s, \boldsymbol{w}), \sigma, (t, \boldsymbol{v})) \in \mu^c$ if and only if the following conditions are satisfied.

- If $k \notin \rho(s)$, then $v_k = w_k$.
- If $k \in \rho(s)$, then $v_k = \sigma$.
- $(s, \{k\}_{\sigma = v_k}, t\} \in \mu$.

The first two conditions state that the input symbol is placed into the windows whose indices belong to $\rho(s)$, and the last condition, in particular, states that the reassignment is made before the automaton changes the current state.

Let $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$ be a word over Σ. A *run* of A on σ consists of a sequence of configurations $c_0, c_1, \ldots, c_n$ such that $c_0 = q_0^c$ and $(c_{i-1}, \sigma_i, c_i) \in \mu^c$, $i = 1, 2, \ldots, n$.

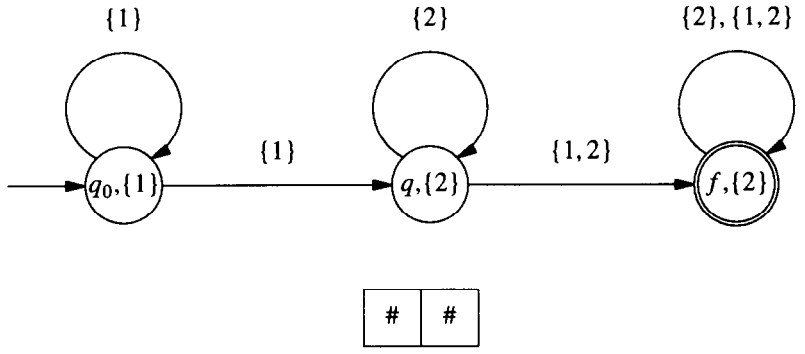
We say that A *accepts* $\sigma \in \Sigma^*$, if there exists a run $c_0, c_1, \ldots, c_n$ of A on σ such that $c_n \in F^c$.

Example 5. Consider an M-memory automaton $A^M = \langle \{q_0, q, f\}, q_0, \#\#, \rho^M, \mu^M, \{f\}\rangle$, where

- $\rho^M(q_0) = \{1\}$, $\rho(q) = \rho(f) = \{2\}$; and
- $\mu^M = \{(q_0, \{1\}, q_0), (q_0, \{1\}, q), (q, \{1, 2\}, f), (q, \{2\}, q), (f, \{1, 2\}, f), (f, \{2\}, f)\}$.

This automaton has the self-explanatory graph representation shown in Fig. 3.

One can easily verify that $L(A^M) = L_1$, where L_1 is the language from Example 1. The behavior of A^M on words where some element of Σ appears twice or more is similar to that of the automaton A from Example 1. Being in the initial state, A^M



initialization

Fig. 3.

stores new input symbols in the first window. When reading a symbol that appears twice, A changes the state to q and, storing new input symbols in the second window, waits for the second appearance of the symbol stored in the first window. Then it enters the final state. For example, $abcbd \in L(A^M)$, and an accepting run of A^M on $abcbd$ is $(q_0, \#\#)$, $(q_0, a\#)$, $(q, b\#)$, (q, bc), (f, bb), (f, bd), compare with Example 1.

Remark 3. Let $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F\rangle$ be an M-automaton. Introducing a new state, if necessary, we may assume that for each $s \in S$ and each $P \subseteq \{1, 2, \ldots, r_A\}$ there exists $t \in S$ such that $(s, P, t) \in \mu$, i.e., A can always make the next move. Indeed, let $q \notin S$. Consider an M-automaton $A' = \langle S \cup \{q\}, q_0, \boldsymbol{u}, \rho', \mu', F\rangle$ such that $\mu' = \mu \cup \{(s, P, q)$: $\{s\} \times \{P\} \times S \cap \mu = \emptyset\} \cup \{q\} \times (2^{\{1,2,\ldots,r_A\}} - \{\emptyset\}) \times \{q\}$; $\rho'(s) = \rho(s)$ for $s \in S$, and $\rho'(q) = \{1, 2, \ldots, r_A\}$. Obviously, A' can always make the next move. Also it cannot leave q, and can enter q if and only if A cannot make the next move. Thus A and A' accept the same language.

Theorem 2. *A language is quasi-regular if and only if it is accepted by an M-automaton.*

Proof. We start with the proof of the "only if" part of the theorem. Let $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F\rangle$ be a finite-memory automaton with r windows. We construct an $(r+1)$-window M-automaton A^M that simulates A. In each stage of the computation r from $(r+1)$ windows of A^M are in one-to-one correspondence with the windows of A, and the only window of A^M that is not in the range of that correspondence contains the input symbol. Thus after the reassignment is made, the input symbol appears either in one or in two windows. In the former case, the input symbol is a "new one" (relatively to the current contents of the windows), and in the latter case the input

symbol already appears in one of the windows of A. This fact will be used by A^M to simulate the behavior of A. In particular, if the input symbol is new, then the correspondence is changed so that the window containing the input symbol corresponds to the window of A containing that symbol (after the reassignment). Otherwise the correspondence remains unchanged.

A formal description of A^M is as follows. Let Π_{r+1} denote the group of all the permutations of $\{1, 2, \ldots, r+1\}$. Then $A^M = \langle S^M, q_0^M, \boldsymbol{u}^M, \rho^M, \mu^M, F^M \rangle$, where

- $S^M = S \times \Pi_{r+1}$. The meaning of a state (s, π) of A^M is as follows. If A^M is in the configuration $((s, \pi), v_1 v_2 \ldots v_{r+1})$, then A is in the configuration $(s, v_{\pi(1)} v_{\pi(2)} \ldots v_{\pi(r)})$.
- $q_0^M = (q_0, \pi_{\mathrm{id}})$, where π_{id} is the identity permutation, i.e., $\pi_{\mathrm{id}}(k) = k$, $k = 1, 2, \ldots, r+1$.
- $\boldsymbol{u}^M = \boldsymbol{u}\#$.
- $\rho^M = (s, \pi) = \{\pi(r+1)\}$, i.e., the input symbol always is placed into the window that does not belong to the range of the permutation of the windows of A.
- $\mu^M = \{((s, \pi), \{\pi(k), \pi(r+1)\}, (t, \pi)) : (s, k, t) \in \mu\} \cup \{((s, \pi), \{\pi(r+1)\}, (t, \pi^{r+1}_{\rho(s)} \pi)) : (s, \rho(s), t) \in \mu\}$, where $\pi^{r+1}_{\rho(s)}$ is the transposition of $\rho(s)$ and $r+1$. That is, $\pi^{r+1}_{\rho(s)}(\rho(s)) = r+1$, $\pi^{r+1}_{\rho(s)}(r+1) = \rho(s)$, and $\pi^{r+1}_{\rho(s)}(k) = k$ for $k \neq \rho(s), r+1$. In accordance with the informal explanation preceding the definition of A^M, the first operand in the union expression for μ^M corresponds to the case when the input symbol appears in the windows of A, and the second operand corresponds to the case when the symbol is new. In both the cases the transition of A^M corresponds to that of A.
- $F^M = F \times \Pi_{r+1}$.

We contend that $L(A) = L(A^M)$. Let $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n \in L(A)$ and let $(s_0, \boldsymbol{w}_0)$, $(s_1, \boldsymbol{w}_1), \ldots, (s_n, \boldsymbol{w}_n)$, be an accepting run of A on σ. We transform it into an accepting run $((s_0, \pi_0), \boldsymbol{v}_0), ((s_1, \pi_1), \boldsymbol{v}_1), \ldots, ((s_n, \pi_n), \boldsymbol{v}_n)$ of A^M on σ by induction as follows. Let $((s_0, \pi_0), \boldsymbol{v}_0) = (q_0^M, \boldsymbol{u}^M)$ and assume that π_i and $\boldsymbol{v}_i$ have been defined so that $v_{i, \pi_i(k)} = w_{i,k}$, $k = 1, 2, \ldots, r$. (Here and hereafter we write $\boldsymbol{w}_i = w_{i,1} w_{i,2} \ldots w_{i,r}$ and $\boldsymbol{v}_i = v_{i,1} v_{i,2} \ldots v_{i,r+1}$.) As usual, we distinguish between the cases of $\sigma_{i+1} \in [\boldsymbol{w}_i]$ and $\sigma_{i+1} \notin [\boldsymbol{w}]$.

Let $\sigma_{i+1} \in [\boldsymbol{w}_i]$. Then for $k \neq r+1$, $v_{i+1, \pi_i(k)} = v_{i, \pi_i(k)}$, and $v_{i+1, \pi_i(r+1)} = \sigma_{i+1}$. Let $\pi_{i+1} = \pi_i$. Since for some $k = 1, 2, \ldots, r$, $\sigma_{i+1} = w_{i,k}$, it follows that $(s_i, k, s_{i+1}) \in \mu$. By the induction hypothesis, $v_{i, \pi_i(k)} = \sigma_{i+1}$, which implies that $((s_i, \pi_i), \{\pi_i(k), \pi_i(r+1)\}, (s_{i+1}, \pi_{i+1}))$ belongs to the first operand in the union expression for μ^M. Thus $(((s_i, \pi_i), \boldsymbol{v}_i), \sigma_{i+1}, ((s_{i+1}, \pi_{i+1}), \boldsymbol{v}_{i+1})) \in \mu^{M\mathrm{c}}$.

Let $\sigma_{i+1} \notin [\boldsymbol{w}_i]$. Then $(s_i, \rho(s_i), s_{i+1}) \in \mu$. The components of v_{i+1} for $k \neq r+1$ are $v_{i+1, \pi_i(k)} = v_{i, \pi_i(k)}$, and $v_{i+1, \pi_i(r+1)} = \sigma_{i+1}$. We put $\pi_{i+1} = \pi^{r+1}_{\rho(s_i)} \pi_i$. Therefore, by the induction hypothesis, for $k \neq \rho(s_i)$, $r+1$, $v_{i+1, \pi_{i+1}(k)} = v_{i, \pi_i(k)} = w_{i,k} = w_{i+1,k}$, and, by the definition of π_{i+1}, $v_{i+1, \pi_{i+1}(\rho(s_i))} = \sigma_{i+1} = w_{i+1, \rho(s_i)}$. Since σ_{i+1} is a new symbol (relatively to $[\boldsymbol{w}_i]$), $(s_i, \rho(s_i), s_{i+1}) \in \mu$. This together with $\{k\}_{v_{i+1,k} = \sigma_{i+1}} = \{\pi_i(r+1)\}$ implies that $((s_i, \pi_i), \{k\}_{v_{i+1,k} = \sigma_{i+1}}, (s_{i+1}, \pi_{i+1}))$ belongs to the second operand in the union expression for μ^M. Thus $(((s_i, \pi_i), \boldsymbol{v}_i), \sigma_{i+1}, (s_{i+1}, \pi_{i+1}), \boldsymbol{v}_{i+1})) \in \mu^{M\mathrm{c}}$, which proves the inclusion $L(A) \subseteq L(A^M)$.

Conversely, let $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n \in L(A^M)$ and let $((s_0, \pi_0), \boldsymbol{v}_0), ((s_1, \pi_1), \boldsymbol{v}_1), \ldots, ((s_n, \pi_n), \boldsymbol{v}_n)$ be an accepting run of A^M on σ. We contend that $(s_0, \boldsymbol{w}_0)$,

$(s_1, \boldsymbol{w}_1), \ldots, (s_n, \boldsymbol{w}_n)$, where $w_{i,k} = v_{i,\pi_i(k)}$, $k = 1, 2, \ldots, r$, $i = 0, 1, \ldots, n$, is an accepting run of $\boldsymbol{A}$ on σ. In order to show that $((s_i, \boldsymbol{w}_i), \sigma_{i+1}, (s_{i+1}, \boldsymbol{w}_{i+1})) \in \mu^c$ we (as usual) distinguish between the cases of $\sigma_i \in [\boldsymbol{w}_i]$ and $\sigma_i \notin [\boldsymbol{w}]$.

Let $\sigma_i \in [\boldsymbol{w}_i]$. Then for some $k \neq r+1$, $\sigma_i = w_{i,k} = v_{i,\pi_i(k)}$. Since $(((s_i, \pi_i), \boldsymbol{v}_i), \sigma_{i+1}, (s_{i+1}, \pi_{i+1}), \boldsymbol{v}_{i+1}) \in \mu^{Mc}$, $((s_i, \pi_i), \{\pi_i(k), \pi_i(r+1)\}, (s_{i+1}, \pi_{i+1})) \in \mu^M$. Therefore $(s_i, k, s_{i+1}) \in \mu$, and $\pi_i = \pi_{i+1}$. This implies $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i$, which, in turn, implies $((s_i, \boldsymbol{w}_i), \sigma_{i+1}(s_{i+1}, \boldsymbol{w}_{i+1})) \in \mu^c$.

Let $\sigma_i \neq [\boldsymbol{w}_i]$. Then, by the induction hypothesis, the only symbol equal to σ_i in $\boldsymbol{v}_{i+1}$ is $v_{i+1,\pi_i(r+1)}$. Since $(((s_i, \pi_i), \boldsymbol{v}_i), \sigma_{i+1}, (s_{i+1}, \pi_{i+1}), \boldsymbol{v}_{i+1})) \in \mu^{Mc}$, $((s_i, \pi_i), \{\pi_i(r+1)\}, (s_{i+1}, \pi_{i+1})) \in \mu^M$. Therefore, by the definition of μ^M, $\pi_{i+1} = \pi^{r+1}_{\rho(s_i)}\pi_i$, and $(s_i, \rho(s_i), s_{i+1}) \in \mu$. This implies $w_{i+1,k} = w_{i,k}$ for $k \neq \rho(s_i)$, and $\boldsymbol{w}_{i+1,\rho(s_i)} = \sigma_i$. Therefore $((s_i, \boldsymbol{w}_i), \sigma_{i+1}(s_{i+1}, \boldsymbol{w}_{i+1})) \in \mu^c$. This completes the proof of the equality $L(\boldsymbol{A}) = L(\boldsymbol{A}^M)$.

Now let $\boldsymbol{A}^M = \langle S^M, q_0^M, \boldsymbol{u}^M, \rho^M, \mu^M, F^M \rangle$ be an M-automaton with r windows. We construct an r-window finite-memory automaton $\boldsymbol{A}$ that stimulates $\boldsymbol{A}^M$. An M-assignment $\boldsymbol{w} = w_1 w_2 \ldots w_r$ of $\boldsymbol{A}^M$ is represented by an assignment $\boldsymbol{v} = v_1 v_2 \ldots v_r$ of $\boldsymbol{A}$ such that $[\boldsymbol{w}] \subseteq [\boldsymbol{v}]$, and a "partition" $p_1, p_2, \ldots, p_r$ of $\{1, 2, \ldots, r\}$ such that $w_l = v_j$ for $l \in p_j$.

A formal description of $\boldsymbol{A} = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F \rangle$ is as follows. Let P_r denote the set of all r-dimensional vectors $\boldsymbol{p} = (p_1, p_2, \ldots, p_r) \in (2^{\{1,\ldots,r\}})^r$ such that $\bigcup_{k=1}^r p_k = \{1, 2, \ldots, r\}$ and $p_i \cap p_j = \emptyset$ for $i \neq j$. Then

- $S = S^M \times P_r$.
- $q_0 = (q_0^M, \boldsymbol{p}^M)$, where $\boldsymbol{p}^M = (p_1^M, p_2^M, \ldots, p_r^M)$ is defined as follows. Let $\boldsymbol{u}^M = u_1^M, u_2^M, \ldots, u_r^M$, and $[\boldsymbol{u}^M] = \{u_1, u_2, \ldots, u_{r'}\}$, where $u_i \neq u_j$ for $i \neq j$. Then, for $k \leqslant r'$, $p_k^M = \{k'\}_{u_{k'}^M = u_k}$, and for $k > r'$, $p_k^M = \emptyset$.
- $\boldsymbol{u} = u_1 u_2 \ldots u_{r'} \#^{r-r'}$, where $u_1, u_2, \ldots, u_{r'}$ are as above.
- $\rho(s, (p_1, p_2, \ldots, p_r))$ is the minimal integer k for which $p_k \subseteq \rho^M(s)$. Since $\rho^M(s) \neq \emptyset$, and either $(p_1, p_2, \ldots, p_r)$ has an empty component, or each of its component is a one-element set, $\rho(s, (p_1, p_2, \ldots, p_r))$ is always defined.
- μ consists of all triples $((s, (p_1, p_2, \ldots, p_r)), k, (s', (p_1', p_2', \ldots, p_r')))$ such that $p_k' = p_k \cup \rho^M(s)$, $p_{k'}' = p_{k'} - \rho^M(s)$, for $k' \neq k$, and $(s, p_k', t) \in \mu^M$. Thus μ reflects μ^M together with ρ^M.
- $F = F^M \times P_r$.

We contend that $L(\boldsymbol{A}) = L(\boldsymbol{A}^M)$. Let $\sigma = \sigma_1 \sigma_2 \cdots \sigma_n \in L(\boldsymbol{A}^M)$ and let $(s_0, \boldsymbol{w}_0), (s_1, \boldsymbol{w}_1), \ldots, (s_n, \boldsymbol{w}_n)$ be an accepting run of $\boldsymbol{A}^M$ on σ. We transform it into an accepting run $((s_0, \boldsymbol{p}_0), \boldsymbol{v}_0), ((s_1, \boldsymbol{p}_1), \boldsymbol{v}_1), \ldots, ((s_n, \boldsymbol{p}_n), \boldsymbol{v}_n)$ of $\boldsymbol{A}$ on σ by induction as follows. Let $((s_0, \boldsymbol{p}_0), \boldsymbol{v}_0) = (q_0, \boldsymbol{u})$ and assume that $\boldsymbol{p}_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,r})$ and $\boldsymbol{v}_i = v_{i,1}, v_{i,2}, \ldots, v_{i,r}$ have been defined such that for each $k = 1, 2, \ldots, r$ and each $l \in p_{i,k}$, $w_{i,l} = v_{i,k}$. Notice that, by the definition of q_0 and $\boldsymbol{u}$, this condition is satisfied for $i = 0$.

Let k be such that $\sigma_{i+1} = v_{i,k}$, if $\sigma_{i+1} \in [\boldsymbol{v}_i]$, and $k = \rho(s_i, \boldsymbol{p}_i)$, if $\sigma_{i+1} \notin [\boldsymbol{v}_i]$. We define $\boldsymbol{p}_{i+1}$ by $p_{i+1,k} = p_{i,k} \cup \rho^M(s_i)$ and $p_{i+1,k'} = p_{i,k'} - \rho^M(s_i)$ for $k' \neq k$, and $\boldsymbol{v}_{i+1}$ by $v_{i+1,k} = \sigma_{i+1}$ and $v_{i+1,k'} = v_{i,k'}$ for $k' \neq k$. Then, since $((s_i, \boldsymbol{w}_i), \sigma_{i+1}, (s_{i+1}, \boldsymbol{w}_{i+1})) \in \mu^{Mc}$, it follows from the definitions of $\boldsymbol{w}_{i+1}$ and $\boldsymbol{v}_{i+1}$ that for each $k = 1, 2, \ldots, r$ and each $l \in p_{i+1,k}$, $w_{i+1,l} = v_{i+1,k}$, i.e., the induction hypothesis is satisfied for $i+1$. Also

$(s_i, p_{i+1,k}, s_{i+1}) \in \mu^M$, which, by the definition of μ, implies $((s_i, \boldsymbol{p}_i), k, (s_{i+1}, \boldsymbol{p}_{i+1})) \in \mu$. Thus $(((s_i, \boldsymbol{p}_i), \boldsymbol{v}_i), \sigma_{i+1}, ((s_{i+1}, \boldsymbol{p}_{i+1}), \boldsymbol{v}_{i+1})) \in \mu^c$, which proves the inclusion $L(A^M) \subseteq L(A)$.

Conversely, let $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n \in L(A^M)$ and let $((s_0, \boldsymbol{p}_0), \boldsymbol{v}_0), ((s_1, \boldsymbol{p}_1), \boldsymbol{v}_1), \ldots, ((s_n, \boldsymbol{p}_n), \boldsymbol{v}_n)$ be an accepting run of A on σ. We contend that $(s_0, \boldsymbol{w}_0), (s_1, \boldsymbol{w}_1), \ldots, (s_n, \boldsymbol{w}_n)$, where $w_{i,l} = v_{i,k}$, for $l \in \rho_{i,k}$ is an accepting run of A^M on σ. By definition, $\boldsymbol{w}_0 = \boldsymbol{u}^M$. We have to prove that for each $i = 0, 1, \ldots, n-1$, $((s_i, \boldsymbol{w}_i), \sigma_{i+1}, (s_{i+1}, \boldsymbol{w}_{i+1})) \in \mu^{Mc}$. Let $\sigma_{i+1} = v_{i+1,k}$. Then $((s_i, \boldsymbol{p}_i), k, (s_{i+1}, \boldsymbol{p}_{i+1})) \in \mu$. By the definition of μ, $p_{i+1,k} = p_{i,k} \cup \rho^M(s_i)$, $p_{i+1,k'} = p_{i,k'} - \rho^M(s_i)$, for $k' \neq k$, and $(s_i, p_{i+1,k}, s_{i+1}) \in \mu^M$. It follows from the definition of $\boldsymbol{w}_i$'s that $w_{i+1,l} = w_{i,l}$ for $l \notin \rho^M(s)$, and $\boldsymbol{w}_{i+1,l} = v_{i+1,k} = \sigma_{i+1}$ for $l \in \rho^M(s_i)$. Thus, by the definition of μ^{Mc}, $((s_i, \boldsymbol{p}_i), \sigma_{i+1}, (s_{i+1}, \boldsymbol{p}_{i+1})) \in \mu^{Mc}$, which completes the proof of the theorem. □

Corollary. *Every quasi-regular language is accepted by a finite-memory automaton whose reassignment function is everywhere defined.*

Proof. Let L be a quasi-regular language. Then, by Theorem 2, $L = L(A^M)$ for some M-automaton A^M. The finite-memory automaton A constructed from A^M in the proof of Theorem 2 satisfies the property required by the corollary. □

Now, in view of Theorem 2, when dealing with closure properties we may consider M-automata.

Theorem 3. *The quasi-regular sets are closed under union, intersection, concatenation and Kleene star.*

Proof. Let $A^1 = \langle S^1, q_0^1, \boldsymbol{u}^1, \rho^1, \mu^1, F^1 \rangle$ and $A^2 = \langle S^2, q_0^2, \boldsymbol{u}^2, \rho^2, \mu^2, F^2 \rangle$ be M-automata, where $\boldsymbol{u}^1$ and $\boldsymbol{u}^2$are words of length r_1 and r_2, respectively.

Closure under union. The proof is based on the standard product construction. By Remark 3, we may assume that A^1 and A^2 can always make the next move. Let $A^\cup = \langle S^1 \times S^2, (q_0^1, q_0^2), \boldsymbol{u}^1\boldsymbol{u}^2, \rho^\times, \mu^\times, (F^1 \times S^2) \cup (S^1 \times F^2) \rangle$ be a M-automaton such that

- $\rho^\times(s^1, s^2) = \rho^1(s^1) \cup \{k + r_1\}_{k \in \rho^2(s^2)}$, i.e., the M-assignment on the first r_1 windows is that of A^1 and the last r_2 windows is that of A^2; and
- $\mu^\times = \{((s^1, s^2), P^1 \cup \{k + r_1 : k \in P^2\}, (t^1, t^2)) : (s^1, P^1, t^1) \in \mu^1$, and $(s^2, P^2, t^2) \in \mu^2\}$.

The automaton $A^\cup$ simultaneously simulates A^1 on the first r_1 symbols of the assignments and A^2 on the last r_2 symbols of the assignments. Thus $L(A^1) \cup L(A^2) = L(A^\cup)$.

Closure under intersection. We use the product construction one more time. Let $A^\cap = \langle S^1 \times S^2, (q_0^1, q_0^2), \boldsymbol{u}^1\boldsymbol{u}^2, \rho^\times, \mu^\times, F^1 \times F^2 \rangle$ be a M-automaton, where $\rho^\times$ and $\mu^\times$ are as above.

Since $A^\cap$ simultaneously simulates A^1 on the first r_1 symbols of the assignments and A^2 on the last r_2 symbols of the assignments, $L(A^1) \cap L(A^2) = L(A^\cap)$.

Closure under concatemation. Renaming the states of A^2, if necessary, we may assume that $S^1 \cap S^2 = \emptyset$. Let $A^{1,2} = \langle S^1 \cup S^2, q_0^1, \boldsymbol{u}^1\boldsymbol{u}^2, \rho^{1,2}, \mu^{1,2}, F^2 \rangle$ be a M-automaton such that

- $\rho^{1,2}(s) = \rho^1(s)$, for $s \in S^1$, and $\rho^{1,2}(s) = \{k + r_1\}_{k \in \rho^2(s)}$, for $s \in S^2$.
- $\mu^{1,2} = \mu' \cup \mu'' \cup \mu'''$, where μ', μ'', and μ''' are defined as follows.
  $\mu' = \{(s, P, t)\colon (s, \{k \in P\colon k \leqslant r_1\}, t) \in \mu^1\}$. This relation is supposed to simulate the behavior of A^1 on a prefix of the input. Thus only the first r_1 windows matter.
  $\mu'' = \{(s, P, t)\colon (s, \{k\colon k + r_1 \in P\}, t) \in \mu^2\}$. This relation is supposed to simulate the behavior of A^2 on a suffix of the input. Thus only the first r_1 windows matter.
  $\mu''' = \{(f, P, s)\colon (q_0^2, (k\colon k + r_1 \in P\}, s) \in \mu^2, f \in F^1\}$. This relation "connects" A^1 to A^2 when passing from a prefix belonging to $L(A^1)$ to a suffix belonging to $L(A^2)$. Thus $A^{1,2}$ first simulates A^1 on the first r_1 windows, and then simulates A^2 on the last r_2 windows.

It can be verified that $L(A^1)(L(A^2) - \{\varepsilon\}) = L(A^{1,2})$, where ε denotes the empty word. Indeed, if $(s_0^1, \boldsymbol{w}_0^1), (s_1^1, \boldsymbol{w}_1^1), \ldots, (s_{n_1}^1, \boldsymbol{w}_{n_1}^1)$ is a run of A^1 on σ_1 and $(s_0^2, \boldsymbol{w}_0^2)$, $(s_2^2, \boldsymbol{w}_2^2), \ldots, (s_{n_2}^2, \boldsymbol{w}_{n_2}^2)$ is a run on A^2 on σ_2 then $(s_0^1, \boldsymbol{w}_0^1\boldsymbol{w}_0^2), (s_1^1, \boldsymbol{w}_1^1\boldsymbol{w}_0^2), \ldots,$ $(s_{n_1}^1, \boldsymbol{w}_{n_1}^1\boldsymbol{w}_0^2), (s_1^2, \boldsymbol{w}_{n_1}^1\boldsymbol{w}_1^2), \ldots, (s_{n_2}^2, \boldsymbol{w}_{n_1}^1\boldsymbol{w}_{n_2}^2)$ is a run of $A^{1,2}$ on $\sigma_1\sigma_2$.

Conversely, let $(s_0, \boldsymbol{w}_0^1\boldsymbol{w}_0^2), (s_1, \boldsymbol{w}_1^1\boldsymbol{w}_1^2), \ldots, (s_n, \boldsymbol{w}_n^1\boldsymbol{w}_n^2)$ be an accepting run of $A^{1,2}$ on $\sigma_1 \ldots \sigma_n$. Then $s_0 = q_0^1$, and $s_n \in F^2$. Since passing from a state of A^1 to a state of A^2 is possible only by means of a transition from μ''', for some $i = 0, 1, \ldots, n-1$, $s_i \in F^1$, $(q_0^2, \{k\colon w_{i+1,k}^2 = \sigma_{i+1}), s_{i+1}) \in \mu^2$, and $\boldsymbol{w}_i^2 = \boldsymbol{u}^2$. Therefore $(s_0, \boldsymbol{w}_0^1), (s_1, \boldsymbol{w}_1^1), \ldots, (s_i, \boldsymbol{w}_i^1)$ is an accepting run of A^1 on $\sigma_1 \ldots \sigma_i$, and $(q_0^2, \boldsymbol{w}_i^2), (s_{i+1}, \boldsymbol{w}_{i+1}^2), \ldots, (s_n, \boldsymbol{w}_n^2)$ is an accepting run of A^2 on $\sigma_{i+1} \ldots \sigma_n$.

Now, if $\varepsilon \notin L(A^2)$, then $L(A^1)L(A^2) = L(A^{1,2})$. Otherwise, $L(A^1)L(A^2) = L(A^{1,2}) \cup L(A^1)$, and the result follows from the closure of quasi-regular languages under union.

Closure under Kleene star. Let $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F \rangle$, where $\boldsymbol{u} = u_1 \ldots u_r$, be an M-automaton. Introducing a new initial state, if necessary, we may assume that $\mu \cap S \times 2^{\{1, \ldots, r\}} \times \{q_0\} = \emptyset$, i.e., A cannot enter q_0 from any of its states. Let $\boldsymbol{u}' = u_1' \ldots u_r'$ be any M-assignment of length r. We intend to prove that a $2r$-window M-automaton $A_{\boldsymbol{u}'} = \langle S \times \{0,1\}^r, (q_0, 0^r), \boldsymbol{u}\boldsymbol{u}', \rho^*, \mu^*, \{(q_0, 0^r)\} \rangle$, where ρ^* and μ^* are defined below, accepts $(L(A))^*$.

- $\rho^*(s, \boldsymbol{a}) = \{k + r\}_{k \in \rho(s)}$. That is, $\rho^*(s)$ is a shift of $\rho(s)$ by r, implying that the first r windows of $A_{\boldsymbol{u}'}$ (which contain the initial M-assignment of A) remain unchanged during the computation. This property of ρ^* is used to reset $A_{\boldsymbol{u}'}$ for processing the coming element of $L(A)$.
- $\mu^* = \mu' \cup \mu''$, where μ' and μ'' are defined as follows.
  μ' consists of the elements $((s, (a_1, a_2, \ldots, a_r)), P, (t, (b_1, b_2, \ldots, b_r)))$ which satisfy the following conditions. If $k \in \rho(s)$, then $b_k = 1$, and if $k \notin \rho(s)$, then $b_k = a_k$. Furthermore for $P_b = \{k\colon k \leqslant r, k \in P$ and $b_k = 0\} \cup \{k\colon r + k \in P$, and $b_k = 1\}$, $(s, P_b, t) \in \mu$.
  $\mu'' = \{((s, \boldsymbol{a}), P, (q_0, 0^r))\colon ((s, \boldsymbol{a}), p, (f, \boldsymbol{b})) \in \mu', (f, \boldsymbol{b}) \in F \times \{0,1\}^r\}$.
  The elements of μ'' "reset" $A_{\boldsymbol{u}'}$ after "accepting" a word from $L(A)$, and the elements of μ' simulate the behavior of A after resetting. Namely, $a_k = 1$ (0) indicates whether

the kth symbol of the M-assignment of A has (not) been replaced, and, respectively, $A_{u'}$ "consults" the second (first) half of its current assignment. That is, the M-assignment $v_1 \dots v_r$ of A corresponding to the M-assignment $w_1 \dots w_{2r}$ of $A_{u'}$ is defined by $v_k = w_k$, if $a_k = 0$, and $v_k = w_{k+r}$, if $a_k = 1$. In other words, $v_k = w_{k+a_k r}$, $k = 1, 2, \dots, r$. Notice, that by the definition of ρ^*, always $w_1 \dots w_r = u$.

We break the proof of the equality $L(A))^* = L(A_{u'})$ into several stages. The first stage is to show that $L(A)$ is a subset of $L(A_{u'})$. Let $\sigma = \sigma_1 \sigma_2 \dots \sigma_n \in L(A)$, and let $(s_0, w_0), (s_1, w_1), \dots, (s_{n-1}, w_{n-1}), (s_n, w_n)$ be an accepting run of A on σ. We transform it into an accepting run of $A_{u'}$ on σ, $((s_0, a_0), uv_0), ((s_1, a_1), uv_1), \dots, ((s_{n-1}, a_{n-1}), uv_{n-1}), ((q_0, 0^r), uv_n)$, where $a_{i,k} = 1$ and if and only if $k \in \bigcup_{j=0}^{i-1} \rho(s_j)$, $i = 0, 1, \dots, n-1$, by induction, as follows.

Let $v_0 = u'$, and assume that v_i have been defined such that $v_{i,k} = w_{i,k}$, if $a_{i,k} = 1$, and $v_{i,k} = u'_k$, if $a_{i,k} = 0$. We define v_{i+1} by $v_{i+1,k} = \sigma_{i+1}$, if $k \in \rho(s_i)$, and $v_{i+1,k} = v_{i+1}$, if $k \notin \rho(s_i)$. It follows from the definition of the a_i's that v_{i+1} satisfies the induction hypothesis for $i+1$. Since $v_{i+1,k}$ is stored in the $(k+r)$th window of $A_{u'}$, our definition of v_{i+1} agrees with the M-reassignment imposed by $\rho^*(s_i)$. Let P denote the set of all indices of the windows of uv_{i+1} containing σ_{i+1}. Then, by the definition of v_{i+1},

$$P = \{k: u_k = \sigma_{i+1}\} \cup \{r+k: v_{i+1,k} (= w_{i+1,k}) = \sigma_{i+1}, a_{i+1,k} = 1\}$$
$$\cup \{r+k: u'_k = \sigma_{i+1}, a_{i+1,k} = 0\}.$$

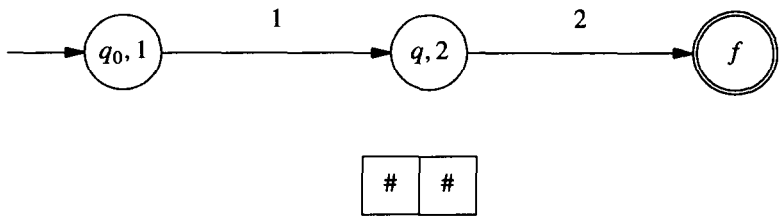
It immediately follows from the definition of a_{i+1} that if $a_{i+1,k} = 0$, then $w_{i+1,k} = u_k$. Therefore

$$P_{a_{i+1}} = \{k: k \leqslant r, k \in P \text{ and } a_{i+1,k} = 0\} \cup \{k: r+k \in P, \text{ and } a_{i+1,k} = 1\}$$
$$= \{k\}_{w_{i+1,k} = \sigma_{i+1}}.$$

Since $(s_i, P_{a_{i+1}}, s_{i+1}) \in \mu$, the membership $(((s_i, a_i), uv_i), \sigma_{i+1}, ((s_{i+1}, a_{i+1}), uv_{i+1})) \in \mu^{*c}$ for $i < n-1$ follows from the definition of μ', and the membership $(((s_{n-1}, a_{n-1}), uv_{n-1}), \sigma_n, ((q_0, 0^r), uv_n)) \in \mu^{*c}$ follows from the definition of μ''.

The second stage of the proof is as follows. Let $\sigma = \sigma_1 \sigma_2 \dots \sigma_n \in L(A_{u'})$, $n > 0$, and let $((q_0, 0^r), uu'), ((s_1, a_1), uv_1), \dots, ((s_{n-1}, a_{n-1}), uv_{n-1}), ((q_0, 0^r), uv_n)$, be an accepting run of $A_{u'}$ on σ such that for $i = 1, 2, \dots, n-1$, $s_i \neq q_0$. Let for $i = 0, 1, \dots, n$, M-assignments w_i be defined by $w_{i,k} = v_{i,k}$, if $a_{i,k} = 1$, and $w_{i,k} = u_k$, if $a_{i,k} = 0$. Let $s_n \in F$ and $a_n \in \{0, 1\}^r$ be such that $(((s_{n-1}, a_{n-1}), uv_{n-1}), \sigma_n, ((s_n, a_n) uv_n) \in \mu$ (see the definition of the μ'' part of μ^*). We contend that $(s_0, w_0), (s_1, w_1), \dots, (s_{n-1}, w_{n-1}), (s_n, w_n)$ is an accepting run of A on σ (implying $\sigma \in L(A)$).

First we examine the relationship between w_i and w_{i+1}. If $k \in \rho(s_i)$, then, by the definition of ρ^*, $k + r \in \rho^*(s_i)$, and, by the definition of μ^*, $a_{i+1,k} = 1$, and $v_{i+1,k} = \sigma_{i+1}$. Thus $w_{i+1,k} (= v_{i+1,k}) = \sigma_{i+1}$. If $k \notin \rho(s_i)$, then $k + r \notin \rho^*(s_i)$, and, by the definition of μ^*, $a_{i+1,k} = a_{i,k}$. Thus $w_{i+1,k} = v_{i+1,k}(u_k) = v_{i,k}(u_k) = w_{i,k}$. This shows that the M-assignments w_i satisfy the requirements on the transition between the configurations of A. We proceed to show that $((s_i, w_i), \sigma_{i+1}, (s_{i+1}, w_{i+1})) \in \mu^c$. Let P denote the set of all



initialization

Fig. 4.

indices of the windows of $A_{u'}$ containing σ_{i+1}, i.e., $P=\{k\}_{u_k=\sigma_i}\cup\{k+r\}_{v_{i+1,k}=\sigma_i}$. Since $w_{i+1,k}=u_k$, if $a_{i+1,k}=0$, and $w_{i+1,k}=v_{i+1,k}$, if $a_{i+1,k}=1$,

$$P_{a_{i+1}}=\{k: k\leqslant r, k\in P \text{ and } a_{i+1,k}=0\}\cup\{k: r+k\in P, \text{ and } a_{i+1,k}=1\}$$
$$=\{k\}_{w_{i+1,k}=\sigma_i}.$$

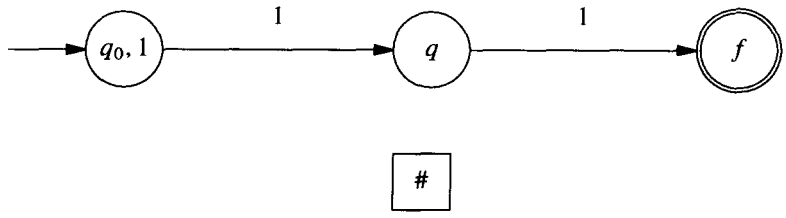
Since $((s_i, \boldsymbol{a}_i), P, (s_{i+1}, \boldsymbol{a}_{i+1}))\in\mu'$, by definition, $(s_i, \boldsymbol{P}_{a_{i+1}}, s_{i+1})\in\mu$, implying $((s_i, \boldsymbol{w}_i), \sigma_{i+1}, (s_{i+1}, \boldsymbol{w}_{i+1}))\in\mu^c$.

Now we are ready to prove the equality $(L(A))^*=L(A_{u'})$. Let $\sigma\in L(A_{u'})$. If σ is the empty word, then, definitely, it belongs to $L(A))^*$. Assume that $\sigma=\sigma_1\ldots\sigma_n$ is not the empty word, and let $((s_0, \boldsymbol{a}_0), \boldsymbol{uv}_0), ((s_1, \boldsymbol{a}_1), \boldsymbol{uv}_1), \ldots, ((s_{n-1}, \boldsymbol{a}_{n-1}), \boldsymbol{uv}_{n-1}), ((s_n, \boldsymbol{a}_n), \boldsymbol{uv}_n)$ be an accepting run of $A_{u'}$ on σ. Let $s_{n_1}, s_{n_2}, \ldots, s_{n_m}$ be all appearances of q_0 in the sequence $s_0, s_1, \ldots, s_n$. (In particular, $n_1=0$ and $n_m=n$.) Then $\sigma_{n_i+1}\ldots\sigma_{n_{i+1}}\in L(A_{\boldsymbol{v}_{n_i}})$, $i=1,2,\ldots,m-1$, and, by the second stage of the proof with $\boldsymbol{u}'=\boldsymbol{v}_{n_i}$, $\sigma_{n_i+1}\ldots\sigma_{n_{i+1}}\in L(A)$. Thus σ is a concatenation of words accepted by A.

Next we prove by induction that any concatenation of the elements of $L(A)$ is accepted by $A_{u'}$. The basis, that is the empty concatenation, is immediate. For the induction step we prove that if $\sigma=\sigma_1\ldots\sigma_n\in L(A_{u'})$, and $\tau=\sigma_{n+1}\ldots\sigma_{n+m}\in L(A)$, then $\sigma\tau\in L(A_{u'})$. Let $((s_0, \boldsymbol{a}_0), \boldsymbol{uv}_0), ((s_1, \boldsymbol{a}_1), \boldsymbol{uv}_1), \ldots, ((s_{n-1}, \boldsymbol{a}_{n-1}), \boldsymbol{uv}_{n-1}), ((s_n, \boldsymbol{a}_n), \boldsymbol{uv}_n)$ be an accepting run of $A_{u'}$ on σ. Thus $(s_n, \boldsymbol{a}_n)=(q_0, 0^r)$. By the first stage of the proof with $\boldsymbol{u}'=\boldsymbol{v}_n$, there is an accepting run $((q_0, 0^r), \boldsymbol{uv}_n), ((s_{n+1}, \boldsymbol{a}_{n+1}), \boldsymbol{uv}_{n+1}), \ldots, (s_{n+m}, \boldsymbol{a}_{n+m}), \boldsymbol{uv}_{n+m})$ of $A_{\boldsymbol{v}_n}$ on τ. Thus $((s_0, \boldsymbol{a}_0), \boldsymbol{uv}_0), (s_1, \boldsymbol{a}_1), \boldsymbol{uv}_1), \ldots, ((s_n, \boldsymbol{a}_n), \boldsymbol{uv}_n)$, $((s_{n+1}, \boldsymbol{a}_{n+1}), \boldsymbol{uv}_{n+1}), \ldots, ((s_{n+m}, \boldsymbol{a}_{n+m}), \boldsymbol{uv}_{n+m})$ is an accepting run of $A_{u'}$ on $\sigma\tau$. □

The following examples show that quasi-regular languages are not closed under either homomorphims or inverse homomorphisms, because in order to accept the (inverse) homomorphic image of a quasi-regular language, a finite-memory automaton may need to "remember" infinitely many identities over the input alphabet.

Example 6. Let $\Sigma=\{\sigma_1\sigma_2, \ldots\}$, $\Sigma'=\{\tau_1\tau_2, \ldots\}$, and $\iota: \Sigma^*\to\Sigma'^*$ be a homomorphism defined by $\iota(\sigma_{3i})=\iota(\sigma_{3i-1})=\tau_{2i}$ and $\iota(\sigma_{3i-2})=\tau_{2i-1}$, $i=1,2,\ldots$ Let A be a finite-memory automaton over Σ defined by the diagram shown in Fig. 4.



initialization

Fig. 5.

Obviously, $L(A)=\{\sigma_i\sigma_j\colon i\neq j\}$ and $\iota(L(A))=\{\tau_i\tau_j\colon i\neq j\}\cup\{\tau_{2i}\tau_{2i}\colon i=1,2,\dots\}$. It has been shown later that $\iota(L(A))$ is not quasi-regular. Thus quasi-regular languages are not closed under homomorphisms. The fact that $\iota(L(A))$ is not quasi-regular can be proved as follows. Assume to the contrary, that it is quasi-regular, and let A' be a finite-memory automaton over Σ' such that $L(A')=\iota(L(A))$. Let i be such that neither τ_{2i} nor τ_{2i+1} belong to $[u_{A'}]$, and let ι' be an automorphism of Σ' that permutes τ_{2i} with τ_{2i+1} and leaves all other symbols unchanged. Since $\tau_{2i}\tau_{2i}\in\iota(L(A))$, by Proposition 2, $\tau_{2i+1}\tau_{2i+1}\in\iota(L(A))(=\{\tau_i\tau_j\colon i\neq j\}\cup\{\tau_{2i}\tau_{2i}\colon i=1,2,\dots\})$. This contradiction completes the proof.

Example 7. Let Σ, Σ' and ι be as in Example 6. Let A' be a finite-memory automaton over Σ' defined by the diagram shown in Fig. 5.

Obviously, $L(A')=\{\tau_i\tau_i\colon i=1,2,\dots\}$ and $\iota^{-1}(L(A'))=\bigcup_{i=1}^{\infty}\{\sigma_i\sigma_i, \sigma_{3i-1}\sigma_{3i}, \sigma_{3i}\sigma_{3i-1}\}$. In order to prove that $\iota^{-1}(L(A'))$ is not quasi-regular, assume to the contrary, that $\iota^{-1}(L(A'))$ is quasi-regular, and let A be a finite-memory automaton over Σ such that $L(A)=\iota^{-1}(L(A'))$. Let i be such that neither σ_{3i-2} nor σ_{3i-1} belong to $[u_A]$, and let ι' be an automorphism of Σ' that permutes τ_{3i-2} with τ_{3i-1} and leaves all other symbols unchanged. Since $\sigma_{3i-1}\sigma_{3i}\in\iota^{-1}(L(A'))$, by Proposition 2, $\sigma_{3i-2}\sigma_{3i}\in\iota^{-1}(L(A'))(=\cup_{i=1}^{\infty}\{\sigma_i\sigma_i, \sigma_{3i-1}\sigma_{3i}, \sigma_{3i}\sigma_{3i-1}\})$. This contradiction implies that $\iota^{-1}(L(A'))$ is not quasi-regular. Thus quasi-regular languages are not closed under inverse homomorphisms.

Remark 4. Actually, under a reasonably weak assumption it can be shown that any class $\boldsymbol{L}$ of languages over an infinite alphabet which is defined by a set of machines having a finite description is not closed under either homomorphisms or inverse homomorphisms. First we observe that, since the set of machines having a finite description is countable, $\boldsymbol{L}$ is countable. We prove that $\boldsymbol{L}$ is not closed under homomorphisms under the assumption that $\Sigma=\{\sigma_1, \sigma_2, \dots\}\in\boldsymbol{L}$. Since $\boldsymbol{L}$ is countable, there exists an infinite subset $L=\{\sigma_{j_1}, \sigma_{j_2}, \dots\}$ of Σ such that $L\notin\boldsymbol{L}$. Let $\iota\colon\Sigma\to\Sigma$ be defined by $\iota(\sigma_i)=\sigma_{j_i}$, $i=1,2,\dots$ Then $\iota(\Sigma)=L$, which shows that $\boldsymbol{L}$ is not closed under homomorphisms.[10]

[10] It follows from the proof that we can replace Σ by any infinite subset.

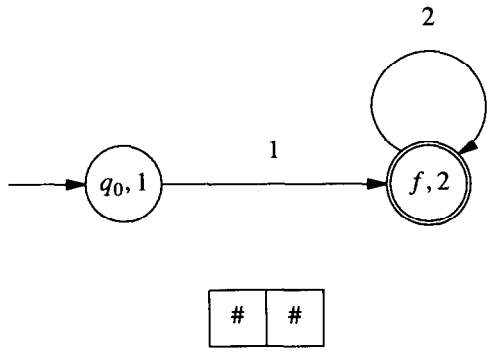


Fig. 6.

Next we prove that $\boldsymbol{L}$ is not closed under inverse homomorphisms under the assumption that $\{\sigma_1\} \in \boldsymbol{L}$. Let $\boldsymbol{L}$ be as above, and let $\iota' : \Sigma \to \Sigma$ be defined by $\iota(\sigma) = \sigma_1$, if $\sigma \in L$; and $\iota(\sigma) = \sigma_2$, otherwise. Then $\iota'^{-1}(\{\sigma_1\}) = L$, which shows that $\boldsymbol{L}$ is not closed under inverse homomorphisms.[11]

Obviously, the above proofs hold for quasi-regular languages as well. However, Examples 6 and 7 are of interest because they are constructive, and the homomorphism in these examples is surjective and finite. That is, its range is all of Σ', and the number of sources of the elements of Σ' is bounded uniformly. Thus quasi-regular language are not closed even under (inverses of) finite homomorphisms.

Finally we show that quasi-regular languages are not closed under reversing.

Example 8. For a word $\sigma = \sigma_1 \sigma_2 \dots \sigma_n$, the *reversal* σ^R of σ is $\sigma^R = \sigma_n \sigma_{n-1} \dots \sigma_1$, and for a language $L \subseteq \Sigma^*$, the reversal L^R of L is $L^R = \{\sigma^R : \sigma \in L\}$. Consider a language L_3 that is defined by

$$L_3 = \{\sigma_1 \sigma_2 \dots \sigma_n : \sigma_i \neq \sigma_1, \ i = 2, 3, \dots, n\}.$$

That is, L_3 consists of all words whose first symbol is different from all other symbols. It could be readily seen that L_3 is accepted by the finite-memory automaton shown in Fig. 6.

Indeed, being in the initial state q_0, the automaton stores the first input symbol in the first window and changes the state to f. Since the automaton cannot leave f and cannot make a move from f on an input stored in the first window, the result follows.

The reversal to L_3 language L_3^R consists of all the words whose last symbol is different from all others. That is, $L_3^R = L_2$, where L_2 is the language from Example 4.

[11] It follows from the proof that we can replace $\{\sigma_1\}$ by any nontrivial subset of Σ, and replace σ_1 and σ_2 by symbols belonging and not belonging, respectively, to that subset.

As was shown in that example, L_2 is not quasi-regular. Thus quasi-regular sets are not closed under reversing.

4. Deterministic and two-way finite-memory automata

In this section we briefly discuss deterministic and two-way deterministic finite-memory automata and present several examples. Some questions which concern the above models and might be of interest are listed in the concluding section. As we mentioned earlier the computation power of these models differs from that of finite-memory automata. First we consider the deterministic one-way model, that is weaker than the nondeterministic one, see Remark 2.

Definition 3. A finite-memory automaton $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F \rangle$ is called *deterministic* if ρ is everywhere defined, and for each $s \in S$ and each $k = 1, 2, \ldots, r_A$ there exists exactly one $t \in S$ such that $(s, k, t) \in \mu$. That is, ρ is a function from S into $\{1, 2, \ldots, r_A\}$ and μ can be thought of as a function from $S \times \{1, 2, \ldots, r_A\}$ into S.

Definition 4. An M-automaton $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F \rangle$ is called *deterministic* if for each $s \in S$, and each nonempty subset W of $\{1, 2, \ldots, r_A\}$ there exists exactly one $t \in S$ such that $(s, W, t) \in \mu$. That is, μ can be thought of as a function from $S \times (2^{\{1, \ldots, r_A\}} - \{\emptyset\})$ into S.

A routine examination of the constructions in Remark 3 and in the proof of Theorem 2 in the previous section shows that a language is accepted by a deterministic finite-memory automaton if and only if it is accepted by a deterministic M-automaton. (This equivalence provides an indirect indication for the robustness of the definition of the deterministic model of computation.) We call the languages accepted by deterministic finite-memory (or M-) automata "deterministic quasi-regular languages." By Remark 2, the languages accepted by deterministic finite-memory automata are closed under complementation, and a straightforward analysis of the proof of the union and intersection parts of Theorem 3 shows that the languages accepted by deterministic M-automata are closed under union and intersection. Therefore the deterministic quasi-regular languages are closed under Boolean operations. However the deterministic quasi-regular languages are not closed under either of reversing, concatenation, and Kleene star. The nonclosure under these operation follows from the examples below.

Example 9. Consider a deterministic finite-memory automaton that has the graph representation shown in Fig. 7.

The automaton behavior is as follows. Being in the initial state, it stores the first input symbols in the first window and changes the state to q. After that, storing new input symbols in the second window, the automaton waits for the second appearance of the symbol stored in the first window. Then it enters the final state which is

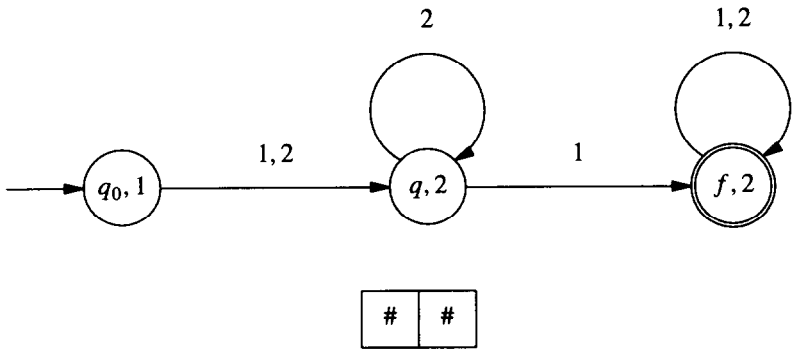


Fig. 7.

impossible to leave. Thus the language L_4 accepted by the above automaton consists exactly of those words where the first symbol appears twice or more:

$$L_4 = \{\sigma_1\sigma_2 \dots \sigma_n\text{: there exist } i = 2, 3, \dots, n \text{ such that } \sigma_i = \sigma_1\}.$$

We have $L_4 = \bar{L}_3$, where L_3 is the language from Example 8. Thus $L_4^R = \bar{L}_3^R = \overline{L_3^R}$. Were the language L_4^R deterministic, by Remark 2, its complement $\bar{L}_4^R = L_3^R$ would also be deterministic, in contradiction with Example 8. Therefore deterministic quasi-regular languages are not closed under reversing.

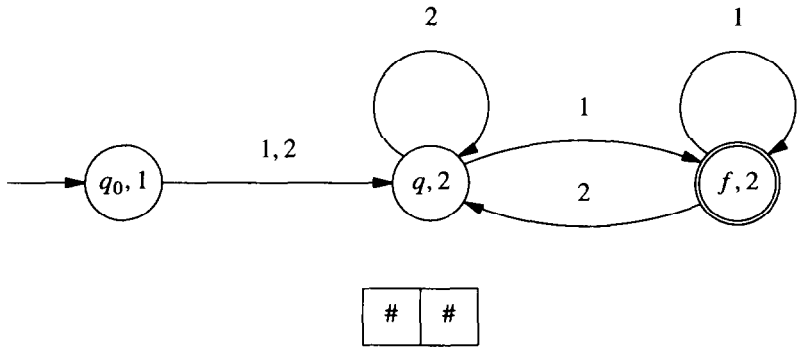
Example 10 below shows that deterministic quasi-regular languages are not closed under either Kleene star or concatenation.

Example 10. Consider a deterministic finite-memory automaton that has the graph representation shown in Fig. 8.

The automaton behavior is as follows. Being in the initial state, it stores the first input symbols in the first window and changes the state to q. The automaton can leave the state q (and enter the only final state f) if and only if the input symbol is equal to that stored in the first window, i.e., to the first input symbol. Furthermore, the automaton can leave f (and enter a nonfinal state q) if and only if the input symbol is not equal to that stored in the first window. Thus the language L_5 accepted by the above automaton consists exactly of the words of the length greater than 1 and with equality holding between the first and the last symbols.

$$L_5 = \{\sigma_1\sigma_2 \dots \sigma_n\text{: } \sigma_1 = \sigma_n,\ n > 1\}.$$

We contend that L_5^* is not a deterministic quasi-regular language. To prove our contention, assume to the contrary that $L_5^* = L(A)$, where $A = \langle S, q_0, \boldsymbol{u}, \rho, \mu, F\rangle$ is a deterministic finite-memory automaton. Let $\sigma_1, \sigma_2, \dots, \sigma_{r_{A+1}}$ be pairwise distinct



initialization

Fig. 8.

elements of Σ. Then for each $i=1,2,\ldots,r_A+1$ the word $\sigma_1\sigma_1\sigma_2\sigma_1\sigma_3\ldots\sigma_1\sigma_{r_A}\sigma_1\sigma_{r_{A+1}}\sigma_i$ belongs to L_5^*, because both $\sigma_1\sigma_1\sigma_2\sigma_1\sigma_3\ldots\sigma_1\sigma_{i-1}\sigma_1$ and $\sigma_i\sigma_1\sigma_{i+1}\sigma_1\sigma_{i+2}\ldots\sigma_1\sigma_r\sigma_1\sigma_{r+1}\sigma_i$ belong to L_5. Since A is a deterministic finite-memory automaton, there is a unique configuration (s, w) that A can enter after reading $\sigma_1\sigma_1\sigma_2\sigma_1\sigma_3\ldots\sigma_1\sigma_{r_A}\sigma_1\sigma_{r_{A+1}}$. Then for each $i=1,2,\ldots,r_A+1$, $A_{(s,w)}=\langle S, s, w, \rho, \mu, F\rangle$ must accept σ_i. Since $A_{(s,w)}$ has r_A windows, there is an $i=1,2,\ldots,r_A+1$ such that $\sigma_i\notin[w]$. Let τ be a symbol different from any of σ_i's and let ι be the automorphism of Σ that permutes τ with σ_i and fixes all other symbols. By Proposition 2, $A_{(s,w)}$ accepts τ. Therefore A accepts the word $\sigma_1\sigma_1\sigma_2\sigma_1\sigma_3\ldots\sigma_1\sigma_r\sigma_1\sigma_{r+1}\tau$, which is impossible, because no suffix of that word belongs to L_5. This shows that deterministic quasi-regular languages are not closed under Kleene star. Notice that it follows from the above proof that the language L_5L_5 is not deterministic either. Thus deterministic quasi-regular language are not closed under concatenation.

Finally we consider two-way deterministic finite-memory automata. Our definition is basically that of a two-way deterministic finite automata, see [8][12], relativized to the case of finite-memory automata.

Definition 5. A *two-way deterministic* finite-memory automaton is a system $A=\langle S, q_0, u, \rho, \mu, F\rangle$, where S, q_0, u, ρ and F are as in a deterministic finite-memory automaton, and the *transition function* μ maps $S\times\{1,2,\ldots,r\}$ into $S\times\{-1,1\}$.

The meaning of μ is as follows. If $\mu(s,k)=(t,-1)$, then in state s, scanning the input symbol stored in the kth window, the automaton enters state t and moves left. If $\mu(s,k)=(t,1)$ then in state s, scanning the input symbol stored in the kth window, the

[12] See also [1, pp. 36–42].

automaton enters state t and moves right. The first and the second components of $\mu(s, k)$ are denoted by $\mu_1(s, k)$ and $\mu_2(s, k)$, respectively. That is, $\mu_1 : S \times \{1, 2, \ldots, r\} \to S$, $\mu_2 : S \times \{1, 2, \ldots, r\} \to \{-1, 1\}$, and $\mu = (\mu_1, \mu_2)$.

A configuration of A is a pair (s, w), where $s \in S$, and w is an assignment of length r. The transition function μ induces the function $\mu^c : S^c \times \Sigma \to S^c \times \{-1, 1\}$ that is defined as follows. Let $c = (s, w)$.

- If σ is the kth symbol of w, then $\mu^c(c, \sigma) = ((\mu_1(s, k), w), \mu_2(s, k))$.
- If $\sigma \notin [w]$, then $\mu^c(c, \sigma) = ((\mu_1(s, \rho(s)), v), \mu_2(s, \rho(s)))$, where v results from w by replacing the $\rho(s)$th symbol of w with σ.

The first and the second components of $\mu^c(s, k)$ are denoted by $\mu_1^c(s, k)$ and $\mu_2^c(s, k)$, respectively.

As in the case of a two-way deterministic finite automaton, the future behavior of a two-way deterministic finite-memory automaton on a given input depends only on the automaton configuration and the head position. The formal definition of this combination is as follows.

Definition 6. Let A be as above. An *instantaneous description* of A on the input word σ is a pair (c, i), where $c \in S^c$ and i is a positive integer not exceeding $|\sigma| + 1$, where $|\sigma|$ denotes the length of σ.

The instantaneous description (c, i) is intended to represent the facts that c is the current automaton configuration during the computation on the input σ, and the automaton head is scanning the ith symbol of σ, if $i \leqslant |\sigma|$, and the head has fallen off the right end of σ, if $i = |\sigma| + 1$.

Next we define the *successor* relation $\vdash_{A,\sigma}$ on the set of instantaneous descriptions of A on σ. Let $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$. If $i = 1$ and $\mu_2^c(c) = -1$, or $i = n + 1$, then (c, i) has no successor. Otherwise $(c, i) \vdash_{A,\sigma} (c', i')$ if and only if $c' = \mu_1^c(c, \sigma_i)$ and $i' = i + \mu_2^c(c, \sigma_i)$. The requirement $i > 1$ for $\mu_2^c(c, \sigma_i) = -1$ prevents any action in the event that the automaton head would move off the left end of the input, and the requirement $i \leqslant n$ prevents any action in the event that the automaton head would move off the right end of the input.

Let $\vdash^*_{A,\sigma}$ be the transitive and reflexive closure of $\vdash_{A,\sigma}$. We say that A *accepts* $\sigma \in \Sigma^*$, if for some final configuration $f^c \in F^c$, $(q_0^c, 1) \vdash_{A,\sigma} (f^c, |\sigma| + 1)$. That is, σ is accepted by A if, starting in the state q_0 and the head on the first symbol of σ, A eventually enters a final state at the same time it falls off the right end of the input. As usual, the set of all words accepted by A is denoted by $L(A)$.

Example 11. It has been shown in Proposition 5 that the complement of the language L_1 presented in Example 1 is not quasi-regular. In this example we show that $\bar{L}_1$ is accepted by a (three-window) two-way deterministic finite-memory automaton. (Recall that $\bar{L}_1$ consists of all words where each symbol appears at most one time.) The reason for $\bar{L}_1$ to be accepted by a two-way deterministic finite-memory automaton is that if the input belongs to $\bar{L}_1$, then it is possible to return to the ith position of the input by remembering σ_i is one of the windows. Before describing formally an

automaton that accepts $\bar{L}_1$, we give a general idea lying behind the proof. Observe that $\sigma_1\sigma_2 \dots \sigma_i \in \bar{L}_1$ if and only if for each $i=2,3,\dots,n$, $\sigma_1\sigma_2 \dots \sigma_i \in \bar{L}_1$. Now, given an input $\sigma=\sigma_1\sigma_2 \dots \sigma_n$, our automaton first stores σ_1 in the first window and then for each $i=2,3,\dots,n$ verifies whether $\sigma_1\sigma_2 \dots \sigma_i \in \bar{L}_1$. For such verification the automaton performs the following sequence of moves. After "accepting" $\sigma_1\sigma_2 \dots \sigma_{i-1}$, the automaton checks whether $\sigma_i=\sigma_1$. If the equality holds, then $\sigma \notin \bar{L}_1$ and the automaton enters a "dead state," i.e., a nonfinal state which is impossible to leave. If $\sigma_i \neq \sigma_1$, the automaton stores σ_i in the second window and starts moving left from σ_i towards σ_1 trying to find out whether there is an $j=2,3,\dots,i-1$ such that $\sigma_j=\sigma_i$. (In this sequence of moves the automaton stores new input symbols in the third window.) If such j exists, then $\sigma \notin \bar{L}_1$ and the automaton enters a dead state. Otherwise the automaton will eventually reach σ_1. Since the automaton already "knows" from the previous verification that $\sigma_1\sigma_2 \dots \sigma_{i-1} \in \bar{L}_1$, arriving to σ_1 indicates that it is at the left end of σ and $\sigma_1\sigma_2 \dots \sigma_i \in \bar{L}_1$. After arriving at the left end of the input, the automaton turns right and moves to σ_i. From σ_i it moves right, enters a final state, and repeats the same procedure starting from σ_{i+1}, etc.

The formal description of a two-way deterministic finite-memory automaton A such that $\bar{L}_1 = L(A)$ is as follows. Let $A=\langle S, q_0, u, \rho, \mu, F\rangle$, where $S=\{q_0, q_1, q_2, q_3, f\}$, $u=\#\#\#$, $\rho(q_0)=1$, $\rho(q_1)=\rho(q_2)=\rho(q_3)=3$, $\rho(f)=2$, $F=\{f\}$, and the transition function μ is defined by the Table 1.

Table 1

	1	2	3
q_0	f, 1	q_1, 1	q_1, 1
q_1	q_1, 1	q_1, 1	q_1, 1
q_2	q_3, 1	q_1, 1	q_2, -1
q_3	q_1, 1	f, 1	q_3, 1
f	q_1, 1	q_2, -1	q_1, 1

A visualized graph representation of A is given by the diagram of Fig. 9. In this diagram the superscript of label k of an edge exiting state s is defined as follows. If $\mu_2(s,k)=-1$, then the superscript is L, and if $\mu_2(s,k)=1$, then the superscript is R. That is, the superscript indicates the direction of the automaton movement.

Let σ be a word over Σ^*. We prove by induction on $|\sigma|$ that if $\sigma \in \bar{L}_1$, then A accepts σ, and if $\sigma \notin \bar{L}_1$, then A running on σ must enter the dead state q_1. The case of $|\sigma|=1,2,3$ can be verified by diagram chasing. In particular, it can be easily seen that if $\sigma_1\sigma_2\sigma_3 \in \bar{L}_1$, then $((q_0, \#\#\#), 1) \vdash^*_{A,\sigma_1\sigma_2\sigma_3} ((f, \sigma_1\sigma_3\sigma_2), 4)$. The reason for considering the case of $n=1,2,3$ separately is to avoid the possibility of $\sigma_{n-1}=\sigma_1$, see the proof of the induction step below.

For the induction step, let $n \geqslant 3$ and assum that for each $\sigma_1\sigma_2 \dots \sigma_n$ if $\sigma_1\sigma_2 \dots \sigma_n \in \bar{L}_1$, then $((q_0, \#\#\#), 1) \vdash^*_{A,\sigma_1\sigma_2 \dots \sigma_n} ((f, \sigma_1\sigma_n\sigma_{n-1}), n+1)$, and if $\sigma_1\sigma_2 \dots \sigma_n \notin \bar{L}_1$, then

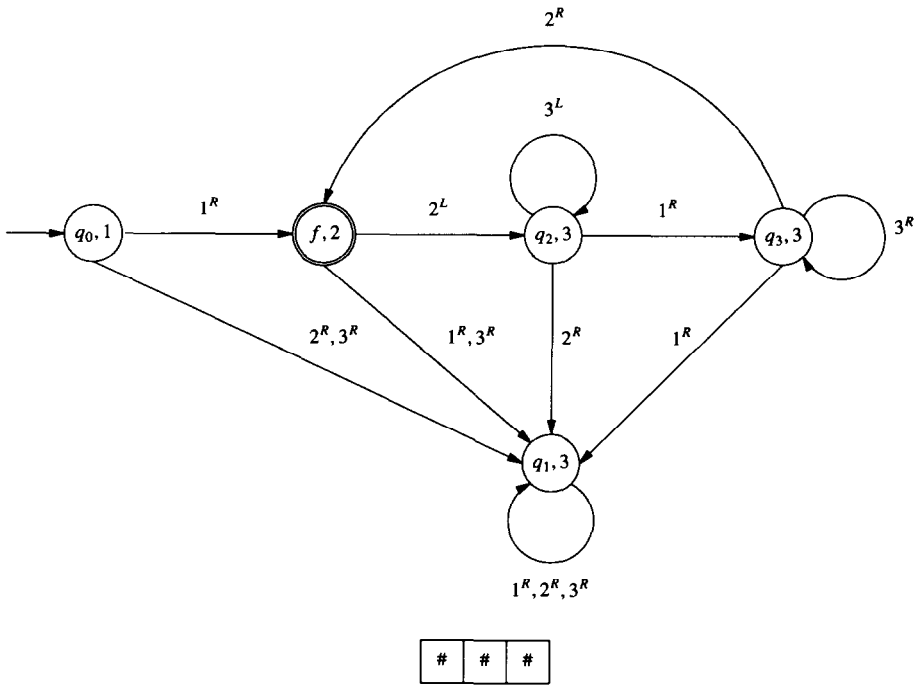


Fig. 9.

A running on $\sigma_1\sigma_2 \dots \sigma_n$ eventually enters q_1. Let $\sigma = \sigma_1\sigma_2 \dots \sigma_{n+1}$. First assume that $\sigma \in \bar{L}_1$. Then $\sigma_1\sigma_2 \dots \sigma_n \in \bar{L}_1$, and, by the induction hypothesis, $((q_0, \#\#\#, 1) \vdash^*_{A, \sigma_1\sigma_2 \dots \sigma_n} ((f, \sigma_1\sigma_n\sigma_{n-1}), n+1)$. Since A is a deterministic automaton, it follows that also $((q_0, \#\#\#, 1) \vdash^*_{A,\sigma} ((f, \sigma_1\sigma_n\sigma_{n-1}), n+1)$. Therefore, in order to prove the induction step for the case of $\sigma \in \bar{L}_1$ it suffices to show that $((f, \sigma_1\sigma_n\sigma_{n-1}), n+1) \vdash^*_{A,\sigma} ((f, \sigma_1\sigma_{n+1}\sigma_n), n+2)$. Since $\sigma_{n+1} \notin [\sigma_1\sigma_n\sigma_{n-1}]$, being in state f with the head scanning σ_{n+1}, the automaton must store it in the second window. Then it has to move left and enter state q_2: $((f, \sigma_1\sigma_n\sigma_{n-1}), n+1) \vdash_{A,\sigma} ((q_2, \sigma_1\sigma_{n+1}\sigma_{n-1}), n)$. Since $\sigma_n \notin [\sigma_n\sigma_{n+1}\sigma_{n-1}]$, the automaton, being in state q_2, must store σ_n in the third window and move left, staying in q_2. After that, in the same manner, A staying in q_2 must continue to move left (storing each symbol it scans in the third window) until it arrives to σ_1:

$$\begin{aligned} ((q_2, \sigma_1\sigma_{n+1}\sigma_{n-1}), n) &\vdash_{A,\sigma} ((q_2, \sigma_1\sigma_{n+1}\sigma_n), n-1) \\ &\vdash_{A,\sigma} ((q_2, \sigma_1\sigma_{n+1}\sigma_{n-1}), n-2) \\ &\vdash_{A,\sigma} \cdots \vdash_{A,\sigma} ((q_2, \sigma_1\sigma_{n+1}\sigma_2), 1). \end{aligned}$$

Then the automaton turns right:

$$((q_3, \sigma_1\sigma_{n+1}\sigma_2), 1) \vdash_{A,\sigma} ((q_3, \sigma_1\sigma_{n+1}\sigma_2), 2)$$
$$\vdash_{A,\sigma} ((q_3, \sigma_1\sigma_{n+1}\sigma_3), 3) \vdash_{A,\sigma} \cdots .$$

Since each symbol A scans differs from its window content, it will arrive to σ_{n+1}, and, finally,

$$((q_3, \sigma_1\sigma_{n+1}\sigma_n), n+1) \vdash_{A,\sigma} ((f, \sigma_1\sigma_{n+1}\sigma_n), n+2).$$

Now assume that $\sigma \notin \bar{L}_1$. If $\sigma_1\sigma_2 \ldots \sigma_n \notin \bar{L}_1$, then, by the induction hypothesis, A enters the dead state before arriving at σ_{n+1}. Otherwise, by the induction hypothesis, $((q_0, \#\#\#), 1) \vdash^*_{A,\sigma} ((f, \sigma_1\sigma_n\sigma_{n-1}), n+1)$. From the instantaneous description $((f, \sigma_1\sigma_n\sigma_{n-1}), n+1)$ the computation of A is as follows. Since $\sigma_1\sigma_2 \ldots \sigma_n \in \bar{L}_1$ and $\sigma_1\sigma_2 \ldots \sigma_n\sigma_{n+1} \notin \bar{L}_1$, for some $i = 1, 2, \ldots, n$, $\sigma_{n+1} = \sigma_i$. If $i = 1, n$, then the automaton immediately enters the dead state. Otherwise, exactly as in the proof of the case of $\sigma \in \bar{L}_1$ it moves left towards σ_1 being in state q_2. When A arrives to σ_i, it enters the dead state, because σ_i is stored in the second window. This completes the proof of the induction step.

5. Concluding remarks and disucssion of future research

In this paper we proposed an extension of the notion of finite automata to finite-memory automata whose inputs are words over finite alphabets and established some basic properties of languages accepted by such automata. It was shown that these languages possess many of the closure and decision properties of ordinary regular languages. Also their restrictions to finite alphabets are regular. Even though finite-memory automata seem to be a quite reasonable model of computation over infinite alphabets, there are no (and cannot be any) formal criteria for accepting them (or any other model) as a *natural* extension of finite automata. Therefore only intuitive arguments and "field tests" can be used to support or reject our definition. Based on some similarity with finite automata, we argued that our model is the "right one." However we do not exclude the possibility that a deeper investigation of languages over infinite alphabets will lead to another more appropriate model. In any case we believe that any simple model of computation over infinite alphabets must be very close to finite-memory automata.

We conclude the paper with some problems which, on one hand, are of interest for their own right, and, on the other hand, might give a better insight into simple languages over infinite alphabets.

- In our opinion, the major problem left unresolved in this paper is whether the containment of quasi-regular languages is decidable. (In [3] we claimed that the inclusion problem $L'' \subseteq L'$ is decidable for any two quasi-regular languages L' and L''. However the proof there is not correct, and we are able to solve the problem

only for the case where L' is accepted by a two-window finite-memory automaton, see the appendix.[13]) A positive answer would imply decidability of the emptiness problem for the languages definable by *systems* of finite-memory automata, see [3]. These languages are interesting for the following reasons. First, they are exactly the boolean closure of the set of the quasi-regular languages. (Thus, in particular, they are closed under Boolean operations.) Second systems of finite-memory automata are analogous to Rabin's automata on infinite words ([6]), and having the decidability of emptiness problem for the languages definable by systems of finite-memory automata, it is very likely that the results of this paper can be extended to "ω-quasi-regular languages."

- The second problem concerns the relationship of deterministic and nondeterministic quasi-regular languages: does each quasi-regular language belong to the closure of deterministic quasi-regular languages under union, intersection, concatenation, and Kleene star?
- The next problem deals with the languages accepted by two-way deterministic finite-memory automata. What closure properties do they possess? Are the emptiness and containment problems for these languages decidable? Is each quasi-regular language accepted by a two-way deterministic finite-memory automaton?
- Finally, how can finite-memory automata be extended to pushdown automata over infinite alphabets? There are two possibilities. The first one is to consider pushdown automata with finite stack alphabets which are able to store input symbols only in their finite memory. The second possibility is to allow storing input symbols both in finite-memory and in the stack. What is the relationship between these two types of pushdown automata? Is it possible to extend the definition of a context-free grammar to infinite alphabets in such way that "quasi-context-free languages" are exactly those accepted by extended pushdown automata?

Appendix A. A decidability result

Here we prove that for a two-window finite-memory automaton A' and for a finite-memory automaton A'' it is decidable whether $L(A'') \subseteq L(A')$. The decision algorithm is as follows. Let $A' = \langle S', q'_0, \boldsymbol{u}', \rho', \mu', F' \rangle$ and $A'' = \langle S'', q''_0, \boldsymbol{u}'', \rho'', \mu'', F'' \rangle$ be a 2- and an r-window finite-memory automaton, respectively. Assume for a moment that we are able to compute a positive integer N (that depends on A' and A'') such that if $L(A'') \not\subseteq L(A')$, then the difference $L(A'') - L(A')$ contains a word σ of length not exceeding N. Then we can proceed as follows.

Let $\tau_1, \tau_2, \ldots, \tau_N$ be pairwise distinct symbols not belonging to $[\boldsymbol{u}'] \cup [\boldsymbol{u}'']$, and let $\Sigma' = \{\tau_i\}_{i=1,\ldots,N} \cup [\boldsymbol{u}'] \cup [\boldsymbol{u}'']$. Since σ contains at most N distinct symbols, there exists an automorphism ι of Σ that is an identity on $[\boldsymbol{u}'] \cup [\boldsymbol{u}'']$ such that $\iota(\sigma)$ is a word over

[13] Even the problem of deciding whether a quasi-regular language is the whole Σ^* seems to us to be very difficult.

the alphabet Σ'. By Proposition 2 (that establishes the closure under automorphisms), $\iota(\sigma)\in L(A'')-L(A')$. Therefore it suffices to check the emptiness of $(L(A'')-L(A'))\cap\Sigma'^*$. Since Σ' is a finite alphabet (its cardinality is at most $N+r+2$), by Proposition 1, the languages $(L(A')\cap\Sigma'^*)$ and $(L(A''))\cap\Sigma'^*)$ are regular. Now the decidability result follows from the equality $(L(A'')-L(A'))\cap\Sigma'^*=(L(A'')\cap\Sigma'^*)-(L(A'))\cap\Sigma'^*)$, the closure of regular languages under boolean operations, and the decidability of the emptiness problem for regular languages. To compute the above constant N we need some preliminary results (four definitions and five lemmas). Below A' and A'' are as in the beginning of the appendix.

With the transition relation μ' we associate a function from $2^{S'^c}\times\Sigma^*$ into $2^{S'^c}$, which we also denote by μ'. (Recall that S'^c denotes the set of all the configurations of A'.) The function μ' is defined by $\mu'(C,\varepsilon)=C$ and $\mu'(C,\sigma\sigma)=\cup_{c\in\mu'(C,\sigma)}\{c'\colon (c,\sigma,c')\in\mu'^c\}$. For example, $\mu'(\{q_0'^c\},\sigma)$ is the set of all the configurations A can enter after reading σ. Thus $L(A')=\{\sigma\colon \mu'(\{q_0'^c\},\sigma)\cap F^c\neq\emptyset\}$. This, in turn, implies that $L(A'')\not\subseteq L(A')$ if and only if $\mu'(\{q_0'^c\},\sigma)\cap F^c=\emptyset\}$, for some $\sigma\in L(A'')$.

This extension of μ' is motivated by the construction of a deterministic finite automaton from a non-deterministic automaton, see [1, Theorem 2.1, pp. 22–23], and will play a similar role. However, in our case, the range of μ', in general, is infinite. This is the reason that some nondeterministic finite-memory automata cannot be converted into deterministic, see Proposition 5 in Section 2.

Next we are going to adapt the classical *product* construction, see [1, pp. 59–60] to the above "deterministic version" of A' and A''. Recall that we are interested in the language $L(A'')\cap\overline{L(A')}$. First we observe that if $(s_0,w_0),(s_1,w_1),\ldots,(s_n,w_n)$ is a run of a finite-memory automaton on a word $\sigma_1\ldots\sigma_n$, then $\sigma_i\in[w_i]$, $i=1,2,\ldots,n$. This invariant property of a run motivates the following two definitions.

Definition A.1. Let $\tau\in\Sigma$. A configuration (s,w) of a finite-memory automaton is called a τ-*configuration* if $\tau\in[w]$.

Definition A.2. An (A',A'')-configuration is a triple (τ,C,c), where $\tau\in\Sigma$, C is a finite set of τ-configurations of A', and c is a τ-configuration of A''.

Now we can define a run of the "product" of A' and A''.

Definition A.3. Let $\sigma=\sigma_1\ldots\sigma_n\in\Sigma^*$. An (A',A'')-*run* on σ is a sequence of (A',A'')-configurations $\bar C_1,\bar C_2,\ldots,\bar C_n$, $\bar C_i=(\sigma_i,C_i,c_i)$, such that $C_{i+1}=\mu'(C_i,\sigma_{i+1})$, and $(c_i,\sigma_{i+1},c_{i+1})\in\mu''^c$, $i=0,1,\ldots,n-1$, where $C_0=\{q_0'^c\}$, $c_0=q_0''^c$.

Our decision algorithm is based on an analysis of (A',A'')-configurations which are the states of the "automaton combined from the complement of A' and A''." First we establish an invariant of (A',A'')-configurations (under automorphisms of Σ). It will be used for computing a positive integer N defined in the first paragraph of the appendix.

Let C be a finite set of τ-configurations of A' and let Σ_C be a subset of $\Sigma \cup \{\#\}$ defined by $\Sigma_C = \{\sigma: (s, \tau\sigma) \in C, \text{ or } (s, \sigma\tau) \in C\}$. Consider a relation $\equiv_\tau$ on Σ_C such that $\sigma_1 \equiv_\tau \sigma_2$ if and only if the following holds. For each $s \in S'$, $(s, \tau\sigma_1) \in C$ if and only if $(s, \tau\sigma_2) \in C$, and $(s, \sigma_1\tau) \in C$ if and only if $(s, \sigma_2\tau) \in C$. It immediately follows from the definition of $\equiv_\tau$ that it is an equivalence relation. The equivalence classes of $\equiv_\tau$ can be described as follows. For each nonempty subset P of $S' \times \{1, 2\}$ define a subset C^P of Σ_C by

$$C^P = \{\sigma \in \Sigma_C: (s, \tau\sigma) \in C \text{ if and only if } (s, 2) \in P, \text{ and}$$

$$(s, \sigma\tau) \in C \text{ if and only if } (s, 1) \in P\}.$$

Then the equivalence classes of $\equiv_\tau$ are in one-to-one correspondence with those subsets P of $S' \times \{1, 2\}$ for which C^P is nonempty.

Let $S'' = \{s_1, s_2, \ldots, s_{l_2}\}$. Let L be the cardinality of $2^{S' \times \{1,2\}} - \{\emptyset\}$, and let $\{P_1, P_2, \ldots, P_L\}$ be an enumeration of $2^{S' \times \{1,2\}} - \{\emptyset\}$. With each (A', A'')-configuration $\bar{C} = (\tau, C, (s_m, w_1, \ldots, w_r))$ we associate an $(2r+2+L)$-dimensional integer vector $V_{\bar{C}} = (n_1, n_2, \ldots, n_{2r+2+L})$ that is defined below.

- For $i = 1, 2, \ldots, 2r$ the component n_i of $V_{\bar{C}}$ is defined as follows. If $w_i = \#$, then $n_{2i-1} = n_{2i} = L+3$. If $w_i = \tau$, then $n_{2i-1} = L+2$ and $n_{2i} = L+4$. If $w_i \notin \Sigma_C \cup \{\#, \tau\}$, then $n_{2i-1} = L+1$ and $n_{2i} = L+5$. Otherwise, for some $j = 1, 2, \ldots, L$, $w_i \in C^{P_j}$, and we put $n_{2i-1} = L-j$ and $n_{2i} = L+5+j$.
- $n_{2r+1} = l_2 - m$, and $n_{2r+2} = l_2 + m$. (Recall that the state part of the third component of $\bar{C}$ is s_m.)
- For $i = 2r+3, \ldots, 2r+2+L$, n_i is equal to the cardinality of C^{P_i}.

The notion of the associated vector reflects the structure of a configuration up to an automorphism of Σ. Namely, let ι be an automorphism of Σ. W can extend ι to configurations by $\iota(s, w) = (s, \iota(w))$,[14] and then to (A', A'')-configurations by $\iota(\tau, C, c) = (\iota(\tau), \{\iota(c')\}_{c' \in C}, \iota(c))$. Then we have the following indistinguishability result.

Lemma A.1. *In the above notation,* $V_{\bar{C}} = V_{\iota(\bar{C})}$.

Proof. It immediately follows from the definition of the extension of ι to configurations, that $\iota(C)$ and $\iota(c)$ are a set of $\iota(\tau)$-configurations and a $\iota(\tau)$-configuration, respectively. Therefore for any $\sigma \in \Sigma_C$, $(s, \tau\sigma) \in C$ if and only if $(s, \iota(\tau)\iota(\sigma)) \in \iota(C)$ and $(s, \sigma\tau) \in C$ if and only if $(s, \iota(\sigma)\iota(\tau)) \in \iota(C)$. The last equivalence implies that for each $P \subseteq S' \times \{1, 2\} - \{\emptyset\}$, $\iota(C^P) = (\iota(C))^P$. Thus the vectors $V_{\bar{C}}$ and $V_{\iota(\bar{C})}$ have the same last L components. The equality of the first $2r+2$ components of the vectors $V_{\bar{C}}$ and $V_{\iota(\bar{C})}$ immediately follows from the fact that the first component of $\iota(\bar{C})$ is $\iota(\tau)$, and the equalities $\iota(\#) = \#$ and $\iota(C^{P_j}) = (\iota(C))^{P_j}$, $j = 1, 2, \ldots, L$, established above. □

[14] Here, as in Lemma 1, we implicitly extend ι to $\Sigma \cup \{\#\}$ by putting $\iota\{\#\} = \#$.

Now we introduce a partial order $\leqslant$ on the set of the associated vectors. Let $V_1=(n_{1,1},\ldots,n_{1,2r+2+L})$ and $V_2=(n_{2,1},\ldots,n_{2,2r+2+L})$. Then $V_1\leqslant V_2$ is and only if $n_{1,i}\leqslant n_{2,i}$, $i=1,2,\ldots,2r+2+L$.

Remark A.1. Let $\bar{C}_1=(\tau_1,C_1,c_1)$, $c_1=(s^1,w_{1,1}\ldots w_{1,r})$ and $\bar{C}_2=(\tau_2,C_2,c_2)$, $c_2=(s^2,w_{2,1}\ldots w_{2,r})$ be (A',A'')-configurations such that $V_{\bar{C}_2}\leqslant V_{\bar{C}_1}$. Then $w_{1,k}=\#$ if and only if $w_{2,k}=\#$, $w_{1,k}=\tau_1$ if and only if $w_{2,k}=\tau_2$, $w_{1,k}\notin\Sigma_{C_1}\cup\{\#,\tau_1\}$ if and only if $w_{2,k}\notin\Sigma_{C_2}\cup\{\#,\tau_2\}$, and $w_{1,k}\in C_1^{P_j}$ if and only if $w_{2,k}\in C_2^{P_j}$. Moreover, $s^1=s^2$. These equalities easily follow from the "double inequality data representation" in the first $2r+2$ components of the associated vectors. Actually, the above equalities are all that we need from the definition of $\leqslant$. However, the double inequality representation is more convenient for a citation of a result from the literature needed for the proof of Lemma A.6 at the end of the appendix.

The nonemptiness of the difference $L(A'')-L(A')$ is tightly related to the notion of *separability* defined below.

Definition A.4. Let $C\subseteq S'^{c}$ be a set of configurations of A', and $c\in S''^{c}$ be a configuration of A''. Let $\sigma\in\Sigma^*$ be a word over Σ. We say that the pair (C,c) is *separated* by σ, if $\sigma\notin\cup_{c'\in C}L(A'_{c'})$ and $\sigma\in L(A''_c)$[15].

In particular, $L(A'')-L(A')$ is nonempty if and only if the pair $(\{q_0'^{c}\},q_0''^{c})$ is separable. (Recall that $q_0'^{c}$ and $q_0''^{c}$ are the initial configurations of A' and A'', respectively.)

There is the following relationship between the separability of (A',A'')-configurations and the partial order on the associated vectors.

Lemma A.2. *Let $\bar{C}_1=(\tau_1,C_1,c_1)$ and $\bar{C}_2=(\tau_2,C_2,c_2)$ be (A',A'')-configurations such that $V_{\bar{C}_2}\leqslant V_{\bar{C}_1}$. If the pair (C_1,c_1) can be separated by a word of length n, then the pair (C_2,c_2) can also be separated by a word of length n.*

Proof. Let $c_1=(s,w_{1,1}\ldots w_{1,r})$ and $c_2=(s,w_{2,1}\ldots w_{2,r})$. (Notice that, by Remark A.1, the state component of c_1 is equal to that of c_2.) Consider the bijection $\iota_1:\{w_{2,1},\ldots,w_{2,r}\}\to\{w_{1,1},\ldots,w_{1,r}\}$ defined by $\iota_1(w_{2,k})=w_{1,k}$, $k=1,2,\ldots,r$. By Remark A.1, ι_1 is well defined. Moreover, $\iota_1(\tau_2)=\tau_1$. Using the inequality $V_{\bar{C}_2}\leqslant V_{\bar{C}_1}$ we can extend ι_1 to an embedding ι_2 of $\Sigma_{\bar{C}_2}\cup\{w_{2,1},\ldots,w_{2,r}\}$ into $\Sigma_{\bar{C}_1}\cup\{w_{1,1},\ldots,w_{1,r}\}$ such that $\iota_2(C_2^{P_j})\subseteq C_1^{P_j}$, $j=1,2,\ldots,L$. Indeed, by Remark A.1, we have the equality $\iota_1(C_2^{P_j}\cap\{w_{2,1},\ldots,w_{2,r}\})=C_1^{P_j}\cap\{w_{1,1},\ldots,w_{1,r}\}$, and, since the cardinality of $C_2^{P_j}$ does not exceed the cardinality of $C_1^{P_j}$, $C_2^{P_j}-\{w_{2,1},\ldots,w_{2,r}\}$ can be embedded into $C_1^{P_j}-\{w_{1,1},\ldots,w_{1,r}\}$.

[15] As in Sections 2 and 3, for a configuration $c=(s,\boldsymbol{w})$ of a finite-memory automaton $A=\langle S,q_0,\boldsymbol{u},\rho,\mu,F\rangle$ we define a finite-memory automaton A_c by $A_c=\langle S,s,\boldsymbol{w},\rho,\mu,F\rangle$. In particular, $Aq_0^c=A$.

Let $\sigma_1 = \sigma_{1,1} \dots \sigma_{1,n}$ be a word of length n that separates (C_1, c_1). Let $\sigma_2 = \sigma_{2,1} \dots \sigma_{2,n}$, where $\sigma_{2,i} = \iota_2^{-1}(\sigma_{1,i})$ if $\sigma_{1,i}$ belongs to the range of ι_2, and $\sigma_{2,i} = \sigma_{1,i}$, if $\sigma_{1,i}$ does not belong to the range of ι_2, $i = 1, 2, \dots, n$. We contend that σ_2 separates (C_2, c_2). To prove our contention we have to show that $\sigma_2 \notin \cup_{c \in C_2} L(A'_c)$ and $\sigma_2 \in L(A''_{c_2})$. Let ι_3 be an automorphism of Σ such that $\iota_3(\sigma_1) = \sigma_2$. (The existence of ι_3 is provided by the definition of σ_2 in the beginning of this paragraph.) Then ι_3 coincides with ι_2^{-1} on the range of ι_2. The membership $\sigma_2 \in L(A''_{c_2})$ immediately follows from Lemma 1 with $\iota = \iota_3$. The proof of the nonmembership $\sigma_2 \notin \cup_{c \in \bar{C}_2} L(A'_c)$ is also easy. If for some $c \in C_2$, $\sigma_2 \in L(A_c)$, then we would have $\sigma_1 \in L(A_{\iota_3^{-1}(c)})$. Since $\iota_3^{-1}(c) \in C_1$, this is impossible. □

Lemma A.3. *Let a positive integer N be such that for every $n \geqslant N$, for every word $\sigma = \sigma_1 \dots \sigma_n$ of length n, and for every (A', A'')-run $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_n$ on σ, the associated sequence $V_{\bar{C}_1}, V_{\bar{C}_2}, \dots, V_{\bar{C}_n}$ contains two vectors $V_{\bar{C}_i}$ and $V_{\bar{C}_j}$, $i < j$, such that $V_{\bar{C}_i} \leqslant V_{\bar{C}_j}$. If the difference $L(A'') - L(A')$ is not empty, then it contains a word shorter than N.*

Proof. Let $\sigma = \sigma_1 \dots \sigma_n$ be a word of the minimum length belonging to the difference $L(A'') - L(A')$. We contend that $n < N$. To prove our contention we assume to the contrary that $n \geqslant N$. Let $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_n$ be an (A', A'')-run on σ, $\bar{C}_i = (\sigma_i, C_i, c_i)$, $i = 1, 2, \dots, n$, such that $c_n \in F''^{c}$. Note that, since $\sigma \notin L(A')$, $C_n \cap F'^{c} = \emptyset$. Let i and j be as in the statement of the lemma. Obviously $\sigma_{j+1} \dots \sigma_n$ separates (C_j, c_j). By Lemma A.2, there exists a word σ' of length $n - j$ that separates (C_i, c_i). Since $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_i$ is an (A', A'')-run on $\sigma_1 \sigma_2 \dots \sigma_i$, the word $\sigma'' = \sigma_1 \dots \sigma_i \sigma'$ separates $(\{q_0'^{c}\}, q_0''^{c})$. Therefore $\sigma'' \in L(A'') - L(A')$. The length of σ'' is $n - j + i < n$, in contradiction with the minimality assumption on the length of σ. □

In order to show how to compute the constant N from Lemma A.3 we need one more auxiliary result.

Lemma A.4. *Let ι be an automorphism of Σ that is an identity on $[u'] \cup [u'']$ and let $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_n$ be an (A', A'')-run on $\sigma = \sigma_1 \sigma_2 \dots \sigma_n$. Then $\iota(\bar{C}_1), \iota(\bar{C}_2), \dots, \iota(\bar{C}_n)$ is an (A', A'')-run on $\iota(\sigma)$.*

Proof. Let $\bar{C}_i = (\sigma_i, C, c_i)$, $i = 1, 2, \dots, n$. By the definition of the extension of ι to (A', A'')-configurations, the first component of $\iota(\bar{C}_i)$ is $\iota(\sigma_i)$, $i = 1, 2, \dots, n$, and it follows from the proof of Lemma 1 that $(\iota(c_i), \iota(\sigma_{i+1}), \iota(c_{i+1})) \in \mu''^{c}$ and $\iota(C_{i+1}) = \mu'(\iota(C_i), \iota(\sigma_{i+1}))$, $i = 0, 1, \dots, n-1$, where $C_0 = \{q_0'^{c}\}$ and $c_0 = q_0''^{c}$. □

Lemma A.5. *Given finite-memory automata A' and A'' one can compute a positive integer N that satisfies the conditions of Lemma A.3.*

Proof. Consider a tree T whose nodes are the empty sequence and the sequences $V_{\bar{C}_1}, V_{\bar{C}_2}, \dots, V_{\bar{C}_n}$, where $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_n$ is an (A', A'')-run on some word of length n. The

sequence $V_{\bar{C}_{1,1}}, V_{\bar{C}_{1,2}}, \ldots, V_{\bar{C}_{1,n_1}}$, is a successor of the sequence $V_{\bar{C}_{2,1}}, V_{\bar{C}_{2,2}}, \ldots, V_{\bar{C}_{2,n_2}}$, if and only if $n_1 = n_2 + 1$ and $V_{\bar{C}_{1,i}} = V_{C_{2,i}}$, for $i = 1, 2, \ldots, n_2$. We intend to prove that T is of a finite branching degree. Moreover, for a given positive integer n we show how to compute the part of T that consists of all vertices of depth not exceeding n. So, given n, let us fix n pairwise distinct symbols $\tau_1, \tau_2, \ldots, \tau_n$ not belonging to $[\boldsymbol{u}'] \cup [\boldsymbol{u}'']$, and let $\Sigma' = \{\tau_i\}_{i=1,\ldots,n} \cup [\boldsymbol{u}'] \cup [\boldsymbol{u}'']$. Let $V_{\bar{C}_1}, V_{\bar{C}_2}, \ldots, V_{\bar{C}_m}$, $m \leqslant n$, be a node of T that results from the $(\boldsymbol{A}', \boldsymbol{A}'')$-run $\bar{C}_1, \bar{C}_2, \ldots, \bar{C}_n$ on $\boldsymbol{\sigma} = \sigma_1, \sigma_2, \ldots, \sigma_m$. There exists an automorphism ι of Σ that is an identity on $[\boldsymbol{u}'] \cup [\boldsymbol{u}'']$ such that $\iota(\boldsymbol{\sigma})$ is a word over the alphabet Σ'. Therefore, by Lemmas A.1 and A.4, the node $V_{\bar{C}_1}, V_{\bar{C}_2}, \ldots, V_{\bar{C}_m}$ can also be obtained from an $(\boldsymbol{A}', \boldsymbol{A}'')$-run on $\iota(\boldsymbol{\sigma})$. That is, for computing the set of all vertices of T of depth not exceeding n we may restrict ourselves to the $(\boldsymbol{A}', \boldsymbol{A}'')$-runs on all words over a finite alphabet Σ'. The number of words over Σ' of length not exceeding n is finite, and for each such word all possible $(\boldsymbol{A}', \boldsymbol{A}'')$-runs on it are computable. Also for any two nodes of T it is decidable whether one is a successor of the other. This completes the description of the algorithm for computing the part of T consisting of all vertices of depth not exceeding n.

Using the above algorithm, for each positive integer N we can check whether each node $V_{\bar{C}_1}, V_{\bar{C}_2}, \ldots, V_{\bar{C}_N}$ of depth N contains two vectors $V_{\bar{C}_i}$ and $V_{\bar{C}_j}$, $i<j$, such that $V_{\bar{C}_i} \leqslant V_{\bar{C}_j}$. Now, if a positive integer N satisfying the conditions of Lemma 4 exists, then it can be found by checking all nodes of depth 1, then all nodes of depth 2, etc. (This process must eventually terminate when we arrive at the right N.)

To complete the proof we have to show that there indeed exists a positive integer N satisfying the conditions of Lemma A.3. Assume to the contrary that there is no such N. That is, for each $N = 1, 2, \ldots$ there exists a node $V_{\bar{C}_{N,1}}, V_{\bar{C}_{N,2}}, \ldots, V_{\bar{C}_{N,N}}$ such that for each i and j, $i<j \leqslant N$, $V_{\bar{C}_{N,i}} \not\leqslant V_{\bar{C}_{N,j}}$. Since the number of such nodes is infinite and T is of a finite branching degree, it follows from the König Infinitary Lemma that there exists an infinite path in T that contains infinitely many nodes $V_{\bar{C}_{N,1}}, V_{\bar{C}_{N,2}}, \ldots, V_{\bar{C}_{N,N}}$ such that for each i and j, $i<j \leqslant N$, $V_{\bar{C}_{N,j}} \not\leqslant V_{\bar{C}_{N,j}}$. These nodes, being on the same path, constitute an infinite subset of prefixes of some infinite sequence $V_{\bar{C}_1}, V_{\bar{C}_2}, \ldots, V_{\bar{C}_n} \ldots$ Therefore the above infinite sequence contains no two vectors $V_{\bar{C}_i}$ and $V_{\bar{C}_j}$, $i<j$, such that $V_{\bar{C}_i} \leqslant V_{\bar{C}_j}$. This contradicts [4, Lemma 4.1] stating that exactly the opposite holds. □

Now, using Lemmas A.3 and A.5, we can implement the algorithm presented in the beginning of the appendix.

References

[1] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).

[2] E.A. Ipser, The infinite state acceptor and its application to AI, *SIGART Newsletter* **107** (1989) 29–31.

[3] M. Kaminski and N. Francez, Finite-memory automata, in: *Proc. 31th Ann. IEEE Symp. on Foundations of Computer Science* (IEEE, New York, 1990) 683–688.
[4] R.M. Karp and R.E. Miller, Parallel program schemata. *J. Comput. System Sci.* **3** (1969) 147–195.
[5] M.O. Rabin, *Automata on infinite objects and Church's problem*, Published for Conference Board of the Mathematical Sciences by the American Mathematical Society No. 13, Providence, R.I., 1972.
[6] M.O. Rabin and D. Scott, Finite automata and their decision problems, *IBM J. Res. Develop.* **3** (1959) 114–125.
[7] Y. Shemesh and N. Francez, Finite-state unification automata and relational languages, *Inform. and Comput.*, to appear.
[8] J.C. Shepherdson, The reduction of two-way automata to one-way automata, *IBM J. Res. Develop.* **3** (1959) 198–200.
[9] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, Volume 1, Chapter 3 (Computer Science Press, Rockville, MD, 1988).
[10] P. Wolper, Expressing interesting properties of programs in propositional temporal logic, in: *Proc. 13th Ann. ACM Symp. on Principles of Programming Languages* (ACM, New York, 1986) 184–193.